The University of Jordan

Authorization Form

I, Ashraf Suyyagh, authorize the University of Jordan to supply copies of my Thesis/ Dissertation to libraries or establishments or individuals on request, according to the University of Jordan regulations.

Signature:

Date: 14/8/2011

الجامعة الأردنيــة
كلية الدراسات العليا

نموذج رقم (١٨)
اقرار والتزام بقوانين الجامعة الأردنية وأنظمتها وتعليماتها لطلبة الماجستير

أنا الطالب: أشرف عماد جبرائيل الصباغ الرقم الجامعي: ٨٠٩٠٤٨٤

التخصص: هندسة الحاسوب الكلية: الهندسة والتكنولوجيا

عنوان الرسالة / الاطروحة

Investigating Synthesis of Efficient Handwritten Arabic Word Recognition on FPGAs.

أعلن بأنني قد التزمت بقوانين الجامعة الأردنية وأنظمتها وتعليماتها وقراراتها السارية المفعول المتعلقة باعداد رسائل الماجستير والدكتوراة عندما قمت شخصيا" باعداد رسالتي / اطروحتي ، وذلك بما ينسجم مع الأمانة العلمية المتعارف عليها في كتابة الرسائل والأطاريح العلمية. كما أنني أعلن بأن رسالتي /اطروحتي هذه غير منقولة أو مستلة من رسائل أو أطاريح أو كتب أو أبحاث أو أي منشورات علمية تم نشرها أو تخزينها في أي وسيلة اعلامية، وتأسيسا" على ما تقدم فإنني أتحمل المسؤولية بأنواعها كافة فيما لو تبين غير ذلك بما فيه حق مجلس العمداء في الجامعة الأردنية بالغاء قرار منحي الدرجة العلمية التي حصلت عليها وسحب شهادة التخرج مني بعد صدورها دون أن يكون لي أي حق في التظلم أو الاعتراض أو الطعن بأي صورة كانت في القرار الصادر عن مجلس العمداء بهذا الصدد.

توقيع الطالب: [signature] أشرف الصباغ التاريخ: ١٤ / ٨ / ٢٠١١

تعتمد كلية الدراسات العليا
هذه النسخة من الرسالة
التوقيع.......... التاريخ ١٨/٨/٢٠١١

INVESTIGATING SYNTHESIS OF EFFICENT HANDWRITTEN ARABIC WORD RECOGNITION ENGINES ON FPGAs

By

Ashraf Emad Suyyagh

Supervisor

Dr. Gheith A. Abandah, Associate Prof.

This Thesis was Submitted in Partial Fulfillment of the Requirements for the Master's Degree of Science in Computer Engineering

Faculty of Graduate Studies

The University of Jordan

August, 2011

تعتمد كلية الدراسات العليا
هذه النسخة من الرسالة
التوقيع......... التاريخ.........

COMMITTEE DECISION

This Thesis/Dissertation (Investigating Synthesis of Efficient Handwritten Arabic Word Recognition Engines on FPGAs) was Successfully Defended and Approved on 8/8/2011

Examination Committee	Signature
Dr. Gheith A. Abandah, (Supervisor) Assoc. Prof. of Computer Engineering	---------------------------
Dr. Mohammad Zaki Khader (Member) Prof. of Electrical Engineering	---------------------------
Dr. Ali H. El-Mousa (Member) Assoc. Prof. of Computer Engineering	---------------------------
Dr. Ali al-Haj (Member) Assoc. Prof. of Computer Engineering Princess Sumaya University for Technology	---------------------------

تعتمد كلية الدراسات العليا
هذه النسخة من الرسالة
التوقيع .... التاريخ ١٠/٨/٢٠١١

INVESTIGATING SYNTHESIS OF EFFICENT HANDWRITTEN ARABIC WORD RECOGNITION ENGINES ON FPGAs

By

Ashraf Emad Suyyagh

Supervisor

Dr. Gheith A. Abandah, Associate Prof.

This Thesis was Submitted in Partial Fulfillment of the Requirements for the Master's Degree of Science in Computer Engineering

Faculty of Graduate Studies

The University of Jordan

August, 2011

COMMITTEE DECISION

This Thesis/Dissertation (Investigating Synthesis of Efficient Handwritten Arabic Word Recognition Engines on FPGAs) was Successfully Defended and Approved on 3/8/2011

Examination Committee	Signature
Dr. Gheith A. Abandah, (Supervisor) Assoc. Prof. of Computer Engineering	------------------------------
Dr. Mohammad Zaki Khader (Member) Prof. of Electrical Engineering	------------------------------
Dr. Ali H. El-Mousa (Member) Assoc. Prof. of Computer Engineering	------------------------------
Dr. Ali al-Haj (Member) Assoc. Prof. of Computer Engineering Princess Sumaya University for Technology	------------------------------

Dedication

I would like to dedicate this work to my loving family for their continuous love, support and encouragement.

A special dedication goes to my friend Enas for standing by my side for the past years.

Acknowledgment

I would like to thank my supervisor Dr. Gheith Abandah for his patience, time and endurance throughout the duration of this research, and for his remarks, insights and understanding, without which I wouldn't have made this contribution.

I am very grateful to Dr. Khalid Darabkeh, the head of the Computer Engineering department for the time he has given me to freely focus on doing this research.

Special thanks go to Eng. Shatha Awawdeh and Eng. Enas Jaara for their cooperation and patience in allowing me to use the labs to carry out my simulations on all labs PCs for weeks!

Many thanks are also due to all the engineers and staff in the Department of Computer Engineering in the University of Jordan for their cooperation, help and understanding throughout my Master studies.

I would like also to thank Eng. Fuad Jamour for his help and hints in regard to the Arabic OCR research.

Special thanks are due to all those unknown people who offered their help, time and experience throughout the internet forums for they helped me overcome many of the design issues in short times.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

ASIC	Application Specific Integrated Circuit
DCT	Discrete Cosine Transform
DFT	Discrete Fourier Transform
DMA	Direct Memory Access
DSP	Digital Signal Processor
FAT	File Allocation Table
FIR	Finite Impulse Response
FPGA	Field Programmable Gate Array
FPU	Floating Point Unit
HAL	Hardware Abstraction Layer
HDL	Hardware Descriptive Language
HMM	Hidden Markov Models
ICDAR	International Conference on Document Analysis and Recognition
IP	Intellectual Property
JTAG	Joint Test Action Group
LE	Logic Element
LKT	Loeve-Karhunen Transform
LUT	Look Up Table
NCM	Normalized Central Moments
NN	Neural Network
OCR	Optical Character Recognition
PE	Processing Elements
PIO	Programmed Input Output
PWL	Piece-Wise Linear Approximation
RCD	Row Column Decomposition
SVM	Support Vector Machines
SOPC	System on Programmable Chip
UART	Universal Asynchronous Receiver Transmitter

INVESTIGATING SYNTHESIS OF HANDWRITTEN ARABIC WORD RECOGNITION ENGINES ON FPGAS

By
Ashraf Suyyagh

Supervisor
Dr. Gheith Abandah

ABSTRACT

Arabic OCR has been an active field of research for years now and still has a lot of room for improvement especially for handwritten Arabic. However, in contrast to other languages, Arabic OCR has limited application domains constrained to few commercial application packages whereas OCR of other languages has been in use in assistive technology devices, office automation and portable applications. Due to the cursive nature and peculiarities of the Arabic script, complex algorithms have been developed; however, this has downgraded the performance in terms of speed affecting application domains. Programmable hardware devices like FPGAs have successfully been employed in accelerating image and video processing applications; they offer the possibility of parallel implementations for many of the image processing and OCR algorithms, thus providing faster applications and low power embedded solutions. In this research, we investigate implementing the feature extraction and word recognition stages of the OCR system into FPGAs; these stages are based on algorithms which will make use of the flexibility and parallelism offered by the hardware platform. We analyze some of the most common feature extraction techniques and report their accuracy rates over a lexicon of 50 words of the IFN/ENIT database, estimate the hardware costs of their suggested parallel implementation, and finally implement the best candidate as well as a neural network based recognition engine. A final total system speed up of 200X is reported over MATLAB implementation without sacrificing accuracy rates and using less than quarter of the logic resources. Future work could utilize the free capacity by developing multiple feature extraction techniques; majority voting could be applied on their classification outputs to further increase overall recognition accuracy.

Chapter 1 - Introduction

Optical Character Recognition (OCR) is the electronic translation of printed or handwritten text into machine encoded electronic format which can be edited, searched within and stored more compactly. OCR research for different languages reached different stages of development and maturity, many researchers reported near perfect accuracy results which led to their application in many consumer products and digitization projects. Nevertheless, actual large scale OCR based projects such as library digitization have reported practical accuracy results ranging from 71% to 98% for printed Latin scripts as an example, this is in contrast to the high recognition rates claimed by the commercial products, and additionally such systems had low speeds (Holley, 2009).

The case for Arabic character and word recognition is no different, research has been active for few decades now, yet few applications have found their way into common use except for very few commercial software applications. These packages are limited to only handling printed Arabic text, Sakhr OCR (Sakhr OCR software webpage, 2011), OmniPage (Omnipage Professional 18 product page, 2011) and ReadIris (ReadIris 12 product page, 2011) are such examples. Though these commercial systems claim near perfect recognition accuracy rates of 99.8% to 96% for high-print and low-print quality respectively, these claims were refuted in the most recent and only comparison study (Kanungo et al, 1999). It was shown that such rates are rarely achieved with recognition rates degraded by more than 20% for non-perfect non-clean input. Despite this result, printed Arabic recognition is not an active topic of research. On the other hand, the case of handwritten Arabic words or text is quite different, it is still an active and important research topic of today where new research results are continuously reported, and since

2005 the performance of systems is compared in a bi-annualy held competition at the *International Conference on Document Analysis and Recognition* (ICDAR). However, due to no complete and highly accurate open vocabulary OCR solution for handwritten Arabic text, no commercial applications have been introduced still (Cheriet, 2008). Recent research results are proclaiming accuracy in the 65% to 99% on limited vocabulary sets. (Margner and El Abed, 2009).

Moreover, research on Arabic OCR has mostly reported recognition accuracy and rarely has it shown system performance metrics such as speed until recently; in the ICDAR 2009 competition, different systems competed against each other in terms of accuracy and average word recognition speed. It was shown that the average time for recognizing a single handwritten Arabic word ranged from ~0.115 seconds to ~17.854 seconds, most systems reporting average times around ~2.5 seconds per word (Margner and El Abed, 2009). No information regarding the platform on which the tests were carried out was disclosed, neither was the implementation language for the competing systems; however, some systems utilized the C-based HTK tool or a modified variant for recognition purposes (Al-Hajj et al, 2009) and (Graves and Schmidhuber, 2009).

As for assessing printed text recognition speed performance, no studies have been conducted; however, (Abdullah and Marsidi, 2008) discuss actual experience and challenges of digitizing the Arabic content of the International Islamic University Malaysia in which Arabic content comprised 30% of library content, using Sakhr OCR and modern Intel Xeon 2.4 GHz processors, they abandoned the content digitization project in favor of only digitizing the title page, abstract and the content pages of Arabic texts, this is due to

the lengthy periods it took with low recognition rates which required immense human review.

Unlike Arabic OCR limited application domains, their non-Arabic counterparts, specifically Latin, have found themselves into new application areas such as assistive technology; for example, driven by recent statistics on learning disabilities, blindness and low vision in the U.S.A, it was shown that almost three million school children have a learning disability, 85% of whom having reading difficulties (U.S. Department of Education**,** 2007) and almost five million people in the US alone are blind or visually impaired (Resnikoff et al, 2004), to this end, in 2010, Intel has recently introduced an assistive solution called the Intel Reader which addresses the needs of this segment of people (Intel Reader Press Kit webpage, 2010) , this Intel-Atom based camera-equipped handheld device allows for the transform of printed English text to the spoken word.

Similar assistive devices could possibly serve the needs of millions of people in the Arab world and other countries where the Arabic script is used. This postulate is based on similar statistics to those of the USA; approximately 16 million people suffer from low vision and blindness in the Middle East Region[*] (Resnikoff et al, 2004), and it is extrapolated that millions suffer from reading disabilities in the Arab world (Elbeheri et al, 2009). For example, studies in Jordan show that ~1% of Jordanian students have a learning disability related to reading and this is amongst the lowest in the Arab world (Ministry of Education, 2010).

Beg et al (2010) suggest that hardware-based implementations of OCR can be up to 100 times faster than software-only methods. It was further suggested that algorithms which

can be efficiently and cost-effectively realized in hardware is needed; this can be achieved by either optimizing the current software algorithms or by developing from the ground up new ones suited for hardware. Unfortunately, researchers direct their efforts on devising and optimizing algorithms which can run efficiently on standard processors, and they neglect approaches which are feasible for hardware implementation due to algorithmic computational complexity (McCready, 2000). Srinivasan et al (2010) have shown that using FPGAs to accelerate preprocessing stage hotpots had a speedup of 9X over optimized multi-threaded software algorithms running on Intel reader with the added advantage of reduced power consumption; work on following OCR stages was suggested to be addressed as future work.

1.1.Motivation

Rice et al, (1996) showed that high word/character recognition rates require complex algorithms which often run at lower speeds, and we have shown in the previous section that the practical application of commercial Arabic OCR solutions did not fulfill their job in large scale digitization projects, this is due to their low processing speeds coupled with their practical low recognition results. Additionally the average word recognition time of Arabic words as of 2009 of ~2.5 seconds (Margner and El Abed,2009) does not meet the timing requirements for its use in assistive technology devices which would help millions of people; to elaborate, the normal rate for intelligible speech is around 155 words per minute (Jones et al, 2007), that is, on average a single word takes 0.387 seconds, such devices are required to recognize, analyze and synthesize text in real time. Yet, the low recognition speed of software based Arabic OCR systems hinders the introduction of such

solutions, for the average word recognition time alone exceeds the timing requirement to deliver synthesized speech at a natural level.

Moreover, no portable Arabic OCR solutions for use in banking operations, postal service automation or office automations are present in contrast to their Latin counterparts (Irispen product page, 2011). Consequently, more complex algorithms are still needed to improve practical recognition accuracy and at the same time perform well in terms of speed. Also due to the accuracy/speed tradeoff, an accelerating platform could offer the speed boost without compromising accuracy, hardware could be such a platform.

Field Programmable Gate Arrays (FPGAs) are configurable logic devices which have successfully been deployed in real time image and video processing offering orders of magnitude acceleration and lower power consumption (Draper et al, 2003). Since OCR and image processing share many algorithms, and that FPGAs allow for the acceleration of many image processing stages, this has given the motivation to explore the feasibility of migrating some of the OCR phases into programmable hardware. Our motivation is further driven by the fact that many of the OCR stages from preprocessing, feature extraction and recognition are parallel in nature, and could be accelerated by using hardware implementations. Moreover, Intel's recent introduction of the Atom E600C processor series, the world's first Atom processor with Altera Integrated FPGAs at low price range of 61$ to 106$ allows for hybrid platforms, in such many of the OCR hotspots can be offloaded to hardware accelerators for better real time processing bringing portable, cost-effective, low-power OCR solutions into reality.

1.2. Objectives

In this thesis we investigate the feasibility of programmable hardware as a platform for Arabic handwritten OCR; the main aim is to analyze some of the most commonly used feature extraction techniques, and to determine which is more efficient in order to implement on hardware; efficiency is measured in terms of recognition accuracy and hardware resources cost. Moreover, we aim for an implementation which would deliver speed up factors of 20X or higher in the final system implementation. In our analysis, we concentrate on the feature extraction stage for it is the core of the OCR problem, and is one of the most complex stages; feature extraction has received the attention of most Arabic Handwritten research due to the fact that finding a set of features which could sufficiently describe and recognize a word is no easy task (Abandah and Anssari, 2009).

1.3.Approach

A small set of 50 words is to be chosen from the IFN/ENIT database, the set size is chosen to serve as a proof of concept, it is further constrained by the hardware resources available in the target configurable hardware platform; initial hardware resource estimates and neural network analysis showed that the target FPGA (Altera cyclone II EP2C70) will only fit for a neural network classifier with 50 output nodes. A higher-end FPGA device could increase the maximum lexicon size. Moreover, our application target domain is a portable solution to recognize words in bank checks or postal services where vocabulary size is limited. In addition, we are addressing the problem using a holistic approach, which works best with small lexicon size. The rationale behind using a holistic approach over a segmentation based one are summarized from (Madhvanath and Govindaraju, 2001):

- “Segmentation based approaches add further algorithmic complexity due to segmentation ambiguity, that is deciding where to segment the word image, and also due to the variability of segment shape; determining the identity of each segment.”
- “In general holistic approaches have an intrinsic advantage circumventing segmentation issues and treating each word as a class unto itself, therefore holistic approaches have the potential to model effects that are unique to the class.”
- “Additionally, holistic features provide information about the word that is clearly orthogonal to the knowledge of characters in it. It stands to reason that the introduction of this knowledge should improve recognition even if the writing is poor.”

Handwritten Arabic is chosen for it is the currently active research topic and is the more general case. We then suggest four feature extraction techniques commonly used in literature and implement their associated MATLAB algorithms using optimized built-in functions whenever possible. The rationale behind our choice of algorithms is discussed in section 3.1.

The steps which we carried out in our research are summarized as follows:

- All the images in the set are thinned and scaled to a fixed size of 64×256 pixels.
- For each algorithm, the features are extracted for all the image samples in the set.
- The features will be fed into the neural network framework of MATLAB.

- Sensitivity analysis is conducted by varying the number of hidden nodes of the neural network, and it will be used to determine best topology for each algorithm based on highest recognition rates.
- We estimate FPGA resource and memory costs of a parallel implementation of the chosen algorithms, such estimation is based on approaches proposed in literature with slight modifications.
- The efficiency of the algorithms is to be calculated, based on which, we recommend the best candidate for hardware implementation.
- The winning feature extraction algorithm as well as the neural network classifier will be implemented on FPGA. An implementation in which the neural network recognition engine would fit and adapt to the limited hardware resources of the FPGA is considered.
- Different variants of the hardware and design decisions are to be implemented and compared to each other.

1.4. Outline

This thesis begins by introducing the necessary background and current literature regarding OCR on FPGAs in Chapter 2. Chapter 3 presents the suggested feature extraction algorithms, their sensitivity analysis, cost estimations and efficiency analysis. System design and implementation is discussed in Chapter 4. Results analysis follows in Chapter 5. The thesis concludes with recommendations and future work.

Chapter 2 – Background and Literature Review

Optical Character Recognition (OCR) is the electronic translation of printed or handwritten text into machine encoded electronic format which can be edited, searched within and stored more compactly, it allows for text reuse and easy access of material. OCR is of significant benefit in human machine interaction as well as office automation. (Lorigo and Govindaraju, 2006). It has been the scope of wide research for many world languages and Arabic is no exclusion, for not only is the Arabic script the official orthography of the Arab world but is also used in many other languages as well, Persian, Kurdish, Malay and Urdu are such examples; thus the target of population is in hundreds of millions. (Khorsheed, 2002).

Decades ago, research began on finding methods best suited for printed Arabic numerals, isolated alphabets and words, this research culminated in introducing commercial software packages to address market needs with claims of near-perfect accuracy, though these figures are most likely unattainable in practice (Kanungo et al, 1999) unless these OCRs are specifically trained to the shapes of the words in the target documents (Saleh et al, 2005), little active research is being done in this domain (Margner and El-Abed, 2009). Research shifted to address the more complicated general Arabic handwritten word and text recognition which is still of great interest due to the room for further improvement.

Handwritten recognition is classified based on how the words are presented to the system into two categories: online and offline (Khorsheed, 2002). In online recognition, the user interacts with handheld devices by writing into an electronic screen by using special stylus, text is recognized as being written in a real time manner, the process of writing is traced

and in every stroke the strength and sequential order of each segment is recorded (Sabaani and El-Sana, 2009) and (Biadsy et al, 2006). On the other hand, offline recognition is based on feeding pre-scanned documents to the system, it is the current research drive due to the nature of Arabic handwriting which further complicates the recognition systems.

Of such complications which offline recognizers must handle and overcome is the cursive nature of the Arabic text, the use of external objects such as diatrics, other shapes such as “Hamza” and “Maddah”, and dots (16 of the 28 Arabic characters have dots which could be positioned either above or below the baseline, dots count vary between one to three), similarities in between character shapes, overlapping of characters and interconnection of neighboring characters. In contrast to Latin characters, Arabic has no upper or lower case categories but instead, Arabic characters have different shapes depending on their position in the word whether at the beginning, middle, final, or standalone, for the shape of the character is dependent on the previous one in the word (Articulation effect.) Figure 1 shows the various shapes Arabic characters can take.

Aside from the specifics above related to Arabic script, the offline recognizers must also cope with the variety of character sizes, the unlimited variations of human handwriting, the variety of writing instruments which affects line thickness, color, and/or stroke quality; thick strokes might cause characters to touch or holes in letters be partially or be completely filled, whereas thin strokes may result in broken characters. Moreover, handwritten recognizers must deal with different slant letters as well as translation problems; for different writers or the same writer might slant their letters differently, or not write the letters in the same position relative to the enclosing borders of the scanned words. (Haraty and Ghaddar, 2004). Therefore, the peculiarities and characteristics of the Arabic

font in particular and handwritten text in general make recognition quite a difficult job and thus various approaches were considered as solutions with varying degrees of success.

Character	Isolated	Initial	Middle	End	Transliteration
Alif	أ	أ	ـا	ـا	*a*
Ba'	ب	بـ	ـبـ	ـب	*b*
Ta'	ت	تـ	ـتـ	ـت	*t*
Tha'	ث	ثـ	ـثـ	ـث	*ṯ*
Jeem	ج	جـ	ـجـ	ـج	*ǧ*
Ḥa'	ح	حـ	ـحـ	ـح	*ḥ*
Kha'	خ	خـ	ـخـ	ـخ	*ḫ*
Dal	د	د	ـد	ـد	*d*
Thal	ذ	ذ	ـذ	ـذ	*ḏ*
Ra'	ر	ر	ـر	ـر	*r*
Zy	ز	ز	ـز	ـز	*z*
Seen	س	سـ	ـسـ	ـس	*s*
Sheen	ش	شـ	ـشـ	ـش	*š*
Sad	ص	صـ	ـصـ	ـص	*ṣ*
Dhad	ض	ضـ	ـضـ	ـض	*ḍ*
T'ah	ط	طـ	ـطـ	ـط	*ṭ*
The'ah	ظ	ظـ	ـظـ	ـظ	*ẓ*
Ain	ع	عـ	ـعـ	ـع	ʿ
Gain	غ	غـ	ـغـ	ـغ	*ġ*
Fa	ف	فـ	ـفـ	ـف	*f*
Qaf	ق	قـ	ـقـ	ـق	*q*
Kaf	ك	كـ	ـكـ	ـك	*k*
Lam	ل	لـ	ـلـ	ـل	*l*
Meem	م	مـ	ـمـ	ـم	*m*
Noon	ن	نـ	ـنـ	ـن	*n*
Ha'	ه	هـ	ـهـ	ـه	*h*
Waw	و	و	ـو	ـو	*w*
Ya	ي	يـ	ـيـ	ـي	*y*

Figure 1 -The Various shapes of Arabic Letters (Khorsheed, 2002)

The OCR problem has been traditionally categorized into two main categories, the segmentation approach and segmentation-free (holistic) approach (Khorsheed, 2002). The

former adds a preprocessing stage of further dividing the word into its constituent letters, extracting the features from each and individually recognizing the characters, the word is then reconstructed from the identified letters. Segmentation can also refer to dividing the word/subword into primitives or symbols which could be a character or fractions of character (Abdelazim and Hashish, 1988). However, these approaches require the further task of determining character boundaries which is no easy one, yet it simplifies the recognition stage afterwards for the cursivness issue is not present anymore. (Khorsheed, 2002)

On the other hand, the segmentation free approach - also known as the holistic or global approach - is based on the identification of the word as a whole unit without the deconstruction and reconstruction to the constituent letters. This approach was inspired by results in cognitive psychology which indicate that humans largely recognize words holistically when reading text. Such approaches gained in popularity as an attractive and more straightforward approach for it avoids the difficulty of character segmentation. It is noteworthy to clarify that even though holistic approaches are known as segmentation-free, the context herein is that no segmentation to the character level is considered, that is no separate characters are recognized and combined, in fact various holistic approaches divide the word into several grids for extracting features within.

These contrasting approaches have different applications, holistic approaches are mostly used when the lexicon size is small as they fail to provide satisfactory results when large lexicon is involved (Burrow, 2004), however they prove themselves useful in many application domains where the number of words is limited, bank checks verification, city names and zip code identification for postal services automation to list a few. Segmentation

based approaches lend themselves useful when unconstrained number of words is used and thus can be used for general applications.

2.1. Overview of OCR (Arabic) Stages

Despite the different paradigms and approaches to the OCR problem, most follow the same processing flow, all word images are preprocessed, and then features are extracted and finally used in the classification and recognition stage. The following subsections further elaborate each of these stages. Figure 2 illustrates the general steps in an OCR system.

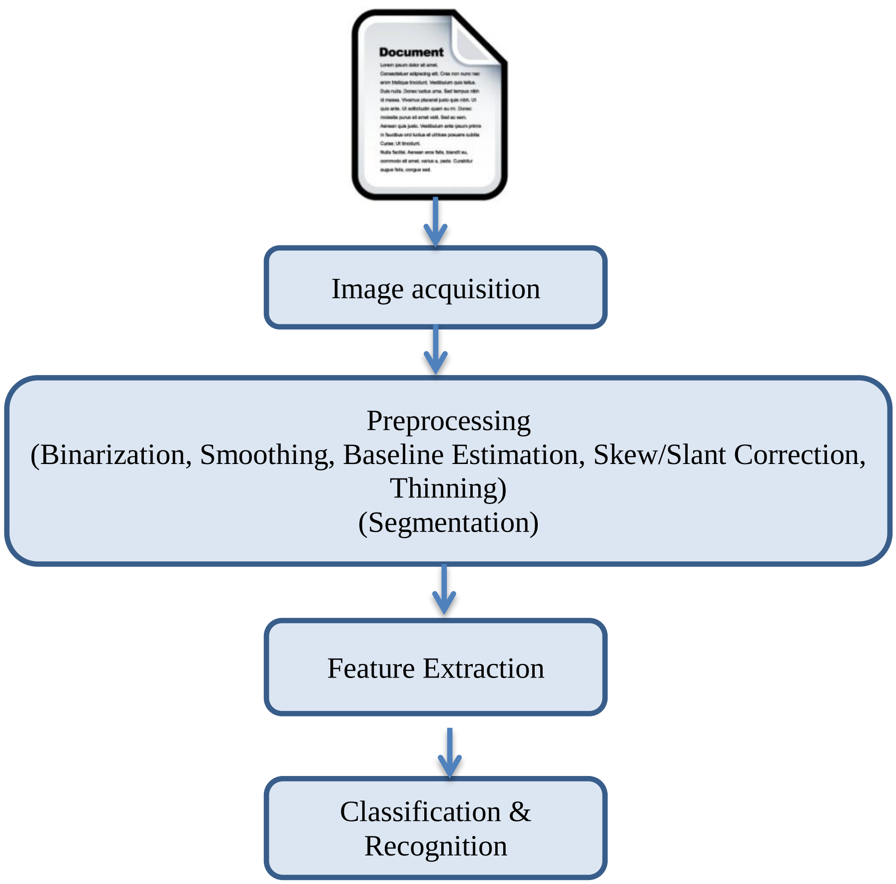


Figure 2 - Major Steps in an OCR System

2.1.1. Preprocessing

The aim of the preprocessing stage is to improve the quality of the images before further processing; this is to compensate for either poor quality originals or poor quality scanning. Scanned images might suffer from the presence of lines, non-character objects, and noise which would introduce errors in subsequent steps and thus need be removed. Moreover, not only is preprocessing concerned with removing spurious segments in the input image, but also ensures consistency of the data presented to the next stages; that is image normalization.

Words vary in sizes, line thickness, quality and slants; depending on the approach, many normalization techniques tend to erode line thickness to one pixel thick representation, known as thinning or skeletonization and different approaches were devised (Jang and Chin, 1992), (Yu and Tsai, 1990), (Larmagnac, 1998) and (Ran et al, 1998), others find the contour of the word, they have the added advantage over skeletonization in that no shape information is lost for thinning algorithms have their ambiguities and might mislocalize features (Lorigo and Govindaraju, 2006), other pre-processing steps include smoothing the image with the intent of noise reduction, and normalizing the slant and skew in handwriting (Khorsheed, 2002). Skew (slope) correction is based on estimating the baseline of the image and its angle of rotation, the baseline is usually determined on the basis of the horizontal histogram projection where the line with the maximum value in the histogram estimates the baseline position, another slightly better baseline estimation is based on using the Hough space (Pechwitz and Maergner, 2003), but the implementation of the projection histogram is far simpler and straightforward and is often used. (Margner and Al-Abed, 2007) Slant correction is used with handwritten scripts or italic printed text, many

algorithms which extract the vertical or near vertical lines have been proposed (Kim and Govindaraju, 1996). Another normalization procedure is binarization, where the grey level of the image is binarized at a certain selected threshold.

2.1.2. Feature Extraction

Feature extraction is defined as the process of acquiring useful information from binarized images in order to be used for classification purposes. The features to be extracted should ideally capture the main characteristics of the character or the word which would make any two different from each other. Consequently handwritten recognition performance largely depends on the feature extraction approach (Haraty and Ghaddar, 2004). The most preferable features are those which are invariant to distortions and variations, and which have the reconstruct-ability property, that is the ability to reconstruct characters and words from their features. Such features are sought in many applications where efficiency and high recognition rates are required (Abandah and Anssari, 2009). In practice, selection of the best features which will give the highest accuracies is a challenging task.

Many features have been proposed in literature for feature extraction, and they are divided into two categories: structural and statistical. Structural features describe the geometric and topological characteristics: local shapes such as edges, crossing points, branching points, curvatures, end points, perceptual features such as loops, ascenders, descenders, directional and topological contour description. Structural methods explicitly and directly capture dot information required to differentiate between letters and thus might be the reason for the common use of structural features in Arabic script (Lorigo and Govindaraju, 2006). They tolerate distortion and variations in writings. However, they are not quite easy to extract (Khorsheed, 2002).

Statistical features are extracted from the statistical distribution of the image pixels and describe characteristics of the input image based on certain measurements; this category includes geometric moment invariants, Hu moments, Zernike moments, wavelet descriptors, Discrete Cosine Transforms (DCT), Discrete Fourier Transforms (DFT), cell intensities (number of pixels representing the word), derivatives of intensity, local slope within a cell and correlation across a window of two cells. (Khorsheed, 2002)

Those features have been widely used in research for Arabic handwritten OCR, and many researchers use a combination of such features for their final feature vector. Pechwitz and Maergner (2003) used a three pixel wide column sliding window in respect to the direction of Arabic handwriting from right to left to capture pixel intensities as features, they used the Loeve-Karhunen Transformation (LKT) on each frame to reduce the vast number of features. Lavrenko et al. (2004) used image properties such as height, width, aspect ratio as well as an estimate of the descenders in the word coupled with discrete Fourier transform as features to use in word recognition in historical documents. Density features were used in (Farah et al, 2005) to identify a lexicon set of 47 Arabic words, the image was divided into 57 variable size grids in which the number of black pixel was summed. El-Hajj et al (2005) used density and concavity features extracted from vertical overlapping frames with a constant width and variable heights. In (Märgner and El-Abed, 2007), a vertical frame split into equal height zones was applied on the word skeleton graph, in each zone frame, lengths of all lines were measured in four directions to form a 20 dimensional normalized feature vector. Rajput and Anita (2010) used DCT along with wavelet transforms in recognizing Indian handwritten script at block level for multi-script identification; they used the features to identify seven types of Indian scripts as well as English text with

accuracy up to 99%. Al-Khateeb et al. (2008) used two dimensional DCT to classify words in a lexicon of 500 Handwritten Arabic words of the IFN/ENIT database. A similar approach was used in (Aburas and Rehiel, 2008) for Arabic letters. Abandah and Anssari (2009) used three sets of 52 Normalized Central Moments (NCM) of orders up to nine and 49 Zernike moments of orders up to 12 to classify Arabic characters, the three sets corresponded to features extracted from the whole letter's image, from the main body and from the secondary components. Zernike moments were further used in (El-Fegh et al, 2009) and (Noaparast and Broumandnia, 2009). Central, Hu and Zernike moments were further investigated in (El-Rube et al, 2010) using different window sizes for printed Arabic sub-word recognition. (Ebrahimpour et al, 2011) used invariant gradient masks to split the thinned word image into four image sets each preserving the vertical, horizontal, left and right diagonals strokes of the word, then density features where extracted from zones around the image center. A similar approach was used in (Su et al, 2009) for Chinese handwritten script.

2.1.3. Classification and Recognition Engines

Classification is defined as the assignment of a certain object or event to one of many predetermined classes, and in OCR context to assign the unknown word to its correct match. Many classifiers have been used in Arabic OCR research for different approaches, *Hidden Markov Models (HMMs)* are statistical models based on a finite set of states each associated with a probability distribution, and the transition in between states is governed by a set of probabilities, in OCR applications, it is assumed that different classes and the feature vector have underlying joint probability. *Support Vector Machines* (SVM) uses kernels to construct linear classification boundaries in high dimension spaces by selecting a

small number of samples of each class to build a linear discriminant function (Abandah and Anssari, 2009). *Decision Trees* divide the feature space into unique regions, and the algorithm need not test every class to find a solution making them effective when large sets are in use, here classification follow a two-step process, initially the input vector is assigned to one of the main groups according to some rules then the input vector is further matched to one of the group members. In *Minimum Distance* classifiers distinct words are characterized by a certain feature vector template and the role of the classifier is to assign the input vector to one of these classes, this assignment is based on using a minimum distance function between the input feature vector and the template, Hamming distance, Euclidian distance and the K-mean are such minimum distance functions. Neural Networks (NN) are nonlinear systems which are characterized by a network topology which is decided by the characteristic of the neurons and the learning methodology. (Hamza, 2008) The reader is referred to (Lorigo and Govindaraju, 2006) and (Korsheed, 2002) for more elaborate discussion of each of the classifiers and their application in various Arabic OCR research, in the subsequent subsection, we will thoroughly elaborate on the classifier in use in this research, neural networks.

2.1.3.1 Neural Networks

An artificial neural network is a nonlinear system which can "learn" to solve complex algorithms from training data sets, the training set consists of a set of pairs of inputs and desired outputs, known as targets (Sarhan and Helalat, 2007), the learning procedure utilizes examples, generalization, associative memory, and fault tolerance; Generalization means that an NN classifies the input data to a desired value, even though input data has not been encountered before. Associative memory creates output even when confronting

unfamiliar or incomplete data. Fault tolerance allows work to continue even when a part of a neuron is faulty or disconnected. (Lee, 2007)

NNs have been applied successfully in many fields including speech recognition, image processing, and adaptive control and is one of the most successful applications that has been proposed for neural networks, this is due to their faster development times, their ability to account of peculiarities of different handwritten styles as well as their parallel architecture, however, they fail to classify newly introduced shapes unless the network is retrained or even its architecture changed. (Korsheed, 2002)

NNs consist of Processing Elements (PEs) called neurons which are interconnected to carry the network function of classification, the neurons are grouped in layers, the input layer, one or more hidden layers and an output layer; the input layer assumes the role of distributing the input feature vector to every node in the first hidden layer, the output layer is the output classes of the network with each neuron corresponding to an output (in our case a word), and the hidden layer determines the mapping ability of the network. One layer of hidden nodes was found to be sufficient for most Arabic OCR applications (Al-Khateeb. 2008) and (Khorsheed, 2002), still, the number of the hidden layers constituent nodes is to be investigated for each problem as it varies depending on the input feature dimensionality and the variance of the samples to be correctly recognized (desired outputs). Increasing the hidden neurons number might reduce the generalization of the network; that is the inability of the network to classify new patterns which are beyond those used in the learning stage; this problem is referred to as over-fitting. (Haraty and Ghaddar, 2004).

Network training is the process by which the parameters of the network, that is the weights, get optimal values with minimum classification errors, one of such methods is called back propagation in which the input data is propagated forward through the network to compute the system output, then the error between the desired and actual output is computed and finally, this error is propagated backward through the network modifying weights on each layer until the first layer is reached. Due to this, these networks are also called “"Recurrent” as they contain feedback connections. This learning ability is the basic feature of intelligence of the neural network as it learns by example, and changes its behavior in response to the environment, and as iterations go by it modifies its weights continuously to match the input set to the desired output. (Hamzah, 2008) Though backpropagation is slow to converge and might get caught in local minima (i.e. not finding the optimal set of weights), it is still simple, straightforward and easy to implement. (Haraty and Ghaddar, 2004).

For every neuron, I_N *and* W_N represent the N^{th} input and associated N^{th} weight of the M^{th} neuron in the network, b_M is an independent bias of the M^{th} neuron. A single neuron assumes the function of producing a weighted linear combination of its inputs, this is done when each of the neuron inputs (I_N) is multiplied by its associated weight (W_N) and all the results are summed, a bias (b_M) independent of the inputs could be added, the results are further applied to a scaling function, more commonly known as a transfer function.

The output of the neuron is defined as

$$x = \sum_{N} I_N W_N + b_M \qquad (1)$$

And the scaled output as:

$$y = f(x) \tag{2}$$

Many types of transfer (scaling) functions are used, they could be linear:

$$y(x) = x \tag{3}$$

Or logarithmic:

$$y(x) = \frac{1}{1 + e^{-x}} \tag{4}$$

Or sigmoid:

$$y(x)\frac{2}{1+ e^{-2x}} - 1 \tag{5}$$

Or hard-limiter:

$$y(x) = \begin{cases} 1 & if\ x < 0 \\ 0 & otherwise \end{cases} \tag{6}$$

The best transfer function to use for a problem is to be investigated empirically.

Figure 3 shows the general structure of a neuron while Figure 4 shows the general architecture of three layered network built from the same neuron.

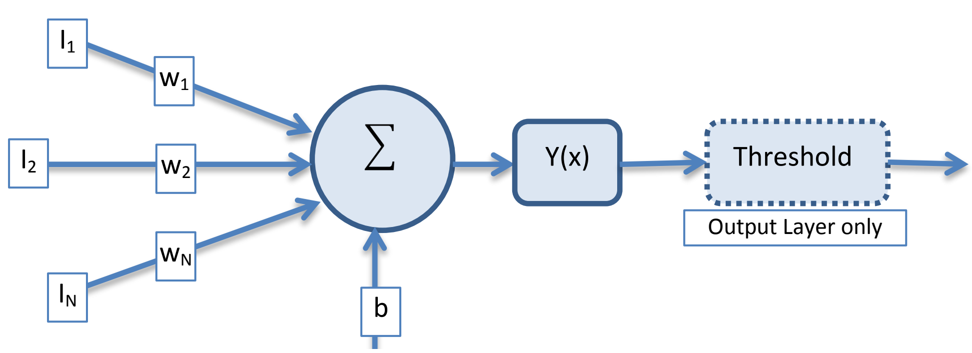


Figure 3 - General Neuron Structure

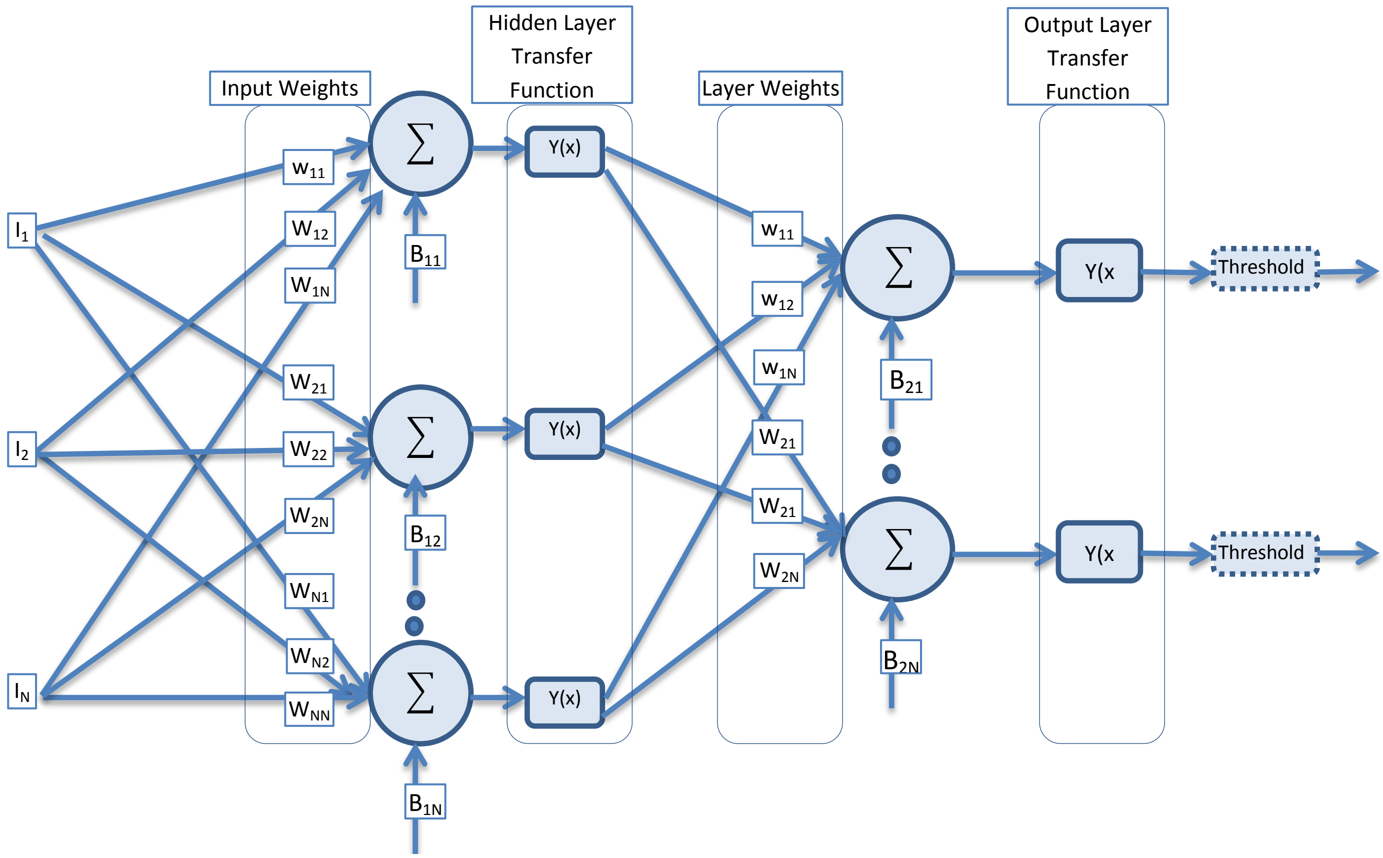


Figure 4- General Neural Network Architecture

2.2. IFN/ENIT Database

The IFN/ENIT-database is a freely available database of Arabic words for non-commercial research. It was developed to advance the research and development of Arabic handwritten word recognition systems and was first introduced at the CIFED02. It was the first attempt to overcome the problem of that large databases are not available to the public, or that researcher often used their own specific datasets which made it quite difficult to compare research results. (Pechwitz and Maergner, 2003)

The database is in version 2.0 patch level 1e (v2.0p1e) and consists of 32492 Arabic words handwritten by more than 1000 writers most of them selected from the narrower range of the Ecole Nationale d’Ingnieurs de Tunis (ENIT). The words set are those of 937 Tunisian

town/village names. Each writer filled one to five forms with preselected town/village names and the corresponding post code. Ground truth was added to the image data automatically and verified manually. In the IFN/ENIT database, words are separated and cropped out during the development stage; they were further de-noised and binarized saving some preprocessing steps. The database is organized in four disjoint sets for the purpose of training and testing. It has been used by more than 30 groups and is the reference database in the ICDAR competition (Margner and El Abed, 2009).

2.3. Field Programmable Gate Array Devices (FPGAs)

An FPGA is a grid of configurable logic fabric that can be used to design and implement digital circuits. FPGA configuration is usually done by means of Hardware Descriptive Languages (HDL) such as VHDL or Verilog. The main advantage of FPGAs over the similar Application Specific Integrated Circuits (ASICs) is the ability to reconfigure the design offering flexibility and non-recurring engineering costs. This advantage offers rapid prototyping capabilities in favor of faster time to market, for designs can be tested and verified in hardware without the need to go through the long fabrication of custom ASIC design, incremental changes and modifications can be implemented in short design cycles, moreover, the availability of high level software tools coupled with intellectual property (IP) cores decreases the learning curve. In addition, FPGAs exceed the computing power of Digital Signal Processors (DSP) through their hardware parallelism achieving higher throughput.(Mirzaei, 2010) For this, FPGAs have become the most popular of Programmable Logic Devices (PLDs) today (Mcready, 2000)

Today's FPGAs not only offer Configurable Logic Blocks (CLB) which include the reprogrammable logic and the mesh of reconfigurable interconnects, but also contain on-chip memories, DSP blocks (i.e. high speed integer multipliers), and some even embedded processors and transceivers offering complete System on a Programmable Chip solutions. (Altera website) Figure 5 shows a typical high level FPGA architecture, that of the Altera Cyclone II is shown. Figure 6 illustrates a Cyclone II logic element structure.

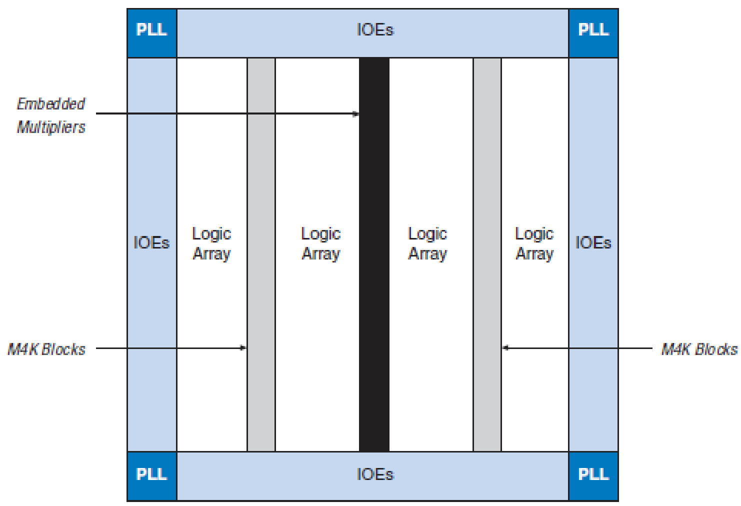


Figure 5 – Typical Modern FPGA Architecture

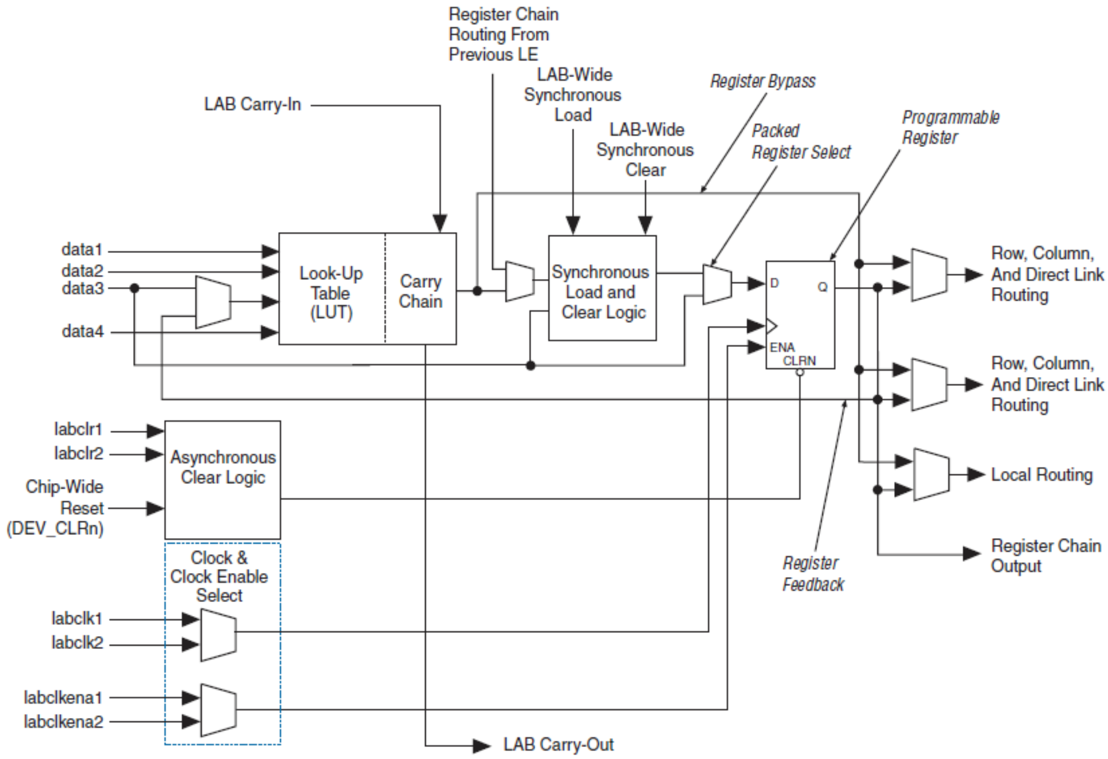


Figure 6 - A Cyclone II Logic Element Internal Structure

2.3.1. FPGAs as Accelerators in Image, Audio and Video Processing

Since FPGAs offer high computational density, a potential for a high level of parallelism, and a pipelined and fine grained nature, they have been the target for image, audio and video processing applications as many require specialized datapaths which can be implemented on FPGAs for real-time applications. They have been employed in object detection problems (Zhao et al, 2009), Face detection (Cho et al, 2009) and (Gao and Lu, 2008), Face recognition (Farugia et al, 2009) and (Timeo et al, 2010), image compression (Agrawal, 2010) and **(**Jyotheswar and Mahapatra, 2007**),** video processing (Bhandari et al, 2010), as well as speech recognition (Fujinaga et al, 2009).

Unfortunately, the complex nature of the algorithms in the image and video processing domain is even exacerbated when implemented in circuit design languages; this has discouraged many researchers from exploiting FPGAs, while the intrepid few who do are repeatedly frustrated by the laborious process of modifying or combining FPGA circuits (Draper et al, 2000). Even though C-to-HDL tools have been introduced to simplify hardware design, they fail to provide results that are close to the hand coded designs. The space constraint is one new challenging limit that is imposed on a C-based tools, furthermore, simple fine grained changes are not easily communicated to the compiler such as adding input or output registers, finally it is extremely difficult to efficiently implement control logic in the pipeline using high level languages. (Mirzaei, 2010)

2.3.2. Neural Networks on FPGAs

FPGAs are suitable hardware platform to implement NN in contrast to ASICs or DSPs in that they preserve the parallel architecture of the network/neuron with flexibility of configuration and modularity, whereas DSPs offer no advantages in their serial execution nor does the hardwired ASIC allow for network topology changes. (Muthuramalingam et al. 2008). However, the multiplication-rich NN algorithm poses quite a challenge for it is quite expensive to implement and thereof a resource / speed tradeoff is a required design choice.

Additionally, since FPGAs are linear devices, it is expensive to realize the non-linear transfer functions of the neuron directly and designers often revert to approximation methods, two practical approaches are used as presented in (Zhu and Sutton, 1999) Piecewise Linear Approximation (PWL) of the nonlinear sigmoid or logsig functions is often used where lines of the form $y = ax+b$ are used to approximate the nonlinear function

with the additional cost of logic resources, yet, should the coefficients *a* and *b* be powers of two, this will reduce the computation to shifts and adds. Look up tables (LUT) are another alternative where samples of the function are taken around its centre and stored in the look up table, usually in on-chip memory. The appropriate number of linear pieces to approximate the nonlinearity of the function and the best number of entries in the LUT (samples) have been explored in (Tommiska, 2003) and (Saichand et al. 2008).

Another issue in regard to NN implementation on FPGAs is data resolution, in his thesis (Lee, 2007) has analyzed the effect of using different floating point units (FPU) on the output of a small neural network, floating point numbers of sizes 32, 24, 20, 16, 14 and 12 were analyzed. It was shown that though a 16 bit FPU saved around 50% or resources, the network will work with 5.91% error rate in comparison with employing FPU32 units, and it was suggested that at least 18-bit FPU be used as a compromise between area and accuracy. In the same work, it was shown that using a fixed point unit of the same bit width as that of the FPU is better than the FPU in terms of accuracy – though not necessarily performance - if the fixed point precision is smaller than the required precision, and only if the range of values is known in advance to account for overflow and underflow problems, this might be difficult if the network is in the learning mode though.

2.3.3. OCR on FPGAs: a Review

Only recently have OCR implementations emerged on hardware platforms, this might have been delayed due to the same reasons presented in section 2.3.1. These hardware implementations have been modest to say the least targeting some preprocessing steps only or full systems at the character recognition level. None of the surveyed systems reported hardware costs or speed up gains however and were only concerned with reporting recognition

rates. (Beg et al, 2008) presented a Verilog model of an artificial neural network for Arabic character recognition, five versions of the 28-character were manually created with slight variations in an 8x8 pixel grid to keep the system within reasonable size, the pixel grid was used as 64 boolean input to the network, and thus no feature extraction stage was used, the network used 28 output neurons, and eight neurons were found to satisfy the accuracy requirement of 95% as reported in software, the system thus had 64 weights per neuron in the hidden layer, all weights were approximated by multiplying the resulting weights from the training stage by 100 and discarding the fractional part, since the inputs were boolean, no multipliers were used but rather tri-state buffers for the weights were controlled by the input pixel values. The hyperbolic tangent transfer function was approximated by means of three linear pieces. Simulation results showed a total accuracy of 80.3%.

Moradi et al. (2010) implemented a system to recognize Farsi handwritten digits, features were extracted from the 40x40 normalized input digit images based on two combined methods which were implemented in parallel, first statistical information about pixel distribution in the upper, lower, right and left halves was used as a feature, the other feature set is based on the number of intersections certain crossing rays have; initially a central horizontal ray is used and the number of intersections it had with the digit is stored, furthermore, the image is divided into eight blocks using three horizontal and three vertical rays and the intersections of each ray is counted, the total number of features of both feature extraction methods is 11 and is used in an 11/16/10 neural network topology with logsig transfer functions, a total accuracy rate of 96% was reported.

Toosizadeh and Eshghi (2005) designed a simple system to recognize single sized font of Persian digits, the digits were written in Microsoft Word, printed, scanned in 300dpi

resolution, resulting digits were scaled and windowed into a 10x7 bitmap matrix. The digits are preprocessed in hardware by removing non-connected pixels after which the system extracted four features: the maximum number of pixels in the horizontal projection, the maximum number of pixels in the vertical projection, the total number of digit pixels of the image and the total number of digit pixels in the top left quarter of the image. A predefined database for the fonts with all features is previously computed and saved. The resulting features are compared to the database; a 0% error rate is claimed.

Razak et al (2007) designed a system for line segmentation for Jawi script (a variant of Arabic). Line segmentation is a preprocessing step which attempts to separate lines from each other in a document before subsequent steps could start. In the authors' proposal, the system loads data into a temporary input memory, followed by a histogram calculation process. If the histogram calculation detects drastic changes in the histogram, it will enable a false minimum elimination function. Elimination process is done until it reaches the point of separation or segmentation point. Following its completion, it will activate a temporary input to accept the segmentation address. Then, it will start to transfer data into the temporary output for line segmentation. Line segmentation of 97% accuracy was reported.

Razak et al. (2009) further continued their work and proposed an algorithm for character segmentation. Histogram normalization and sliding windows are used for hardware implementation of real-time off-line handwritten Jawi script character segmentation with character segmentation accuracy of 98%.

Ahmadi et al (2005) designed an FPGA based system to recognize the printed 26 letters of the English language, the system had a noise removal block by means of using a Finite

Impluse Response (FIR), the characters are segmented from the scanned words and normalized into 16x16 windows, scaling is done using a bilinear interpolation algorithm. Character classification is carried out by a nearest distance search algorithm applying the associative memory; the normalized segmented character is matched as a 256 bit vector to a number of reference patters using the Manhatthen distance measure and the reference pattern with the minimum distance is consider the winner class. 99.5% recognition rate is reported.

Another FPGA based system for printed English alphabet was presented in (Dang'ana, 2008), the system reads a 24 bit image containing characters to be translated, mapping out the locations of the characters, applying binarization, reading the character pixels, classification is done based on template matching and finally the pixel block is converted into an ASCII equivalent

2.4. Development Environment and Target Board

To carry out the investigation in this research, many software programs and utilities were used. The batch resize tool in ACDsee Pro version 4 was used for scaling the image set to the fixed size of 64×256 pixels. The batch rename in the same program was used for renaming the image set into sequential numbers to facilitate image access and processing. The 32-bit version of MATLAB R2009a ver. 7.8.0.347 (Mathworks website, 2011) was the main software platform for algorithm implementation; neural network code was generated using the neural network pattern recognition tool "nprtool", this tool is available from the neural network toolbox in MATLAB. All software was installed on a fresh installation of a 64-bit Windows 7 Ultimate version running on a system equipped with an Intel core T6600 processor at 2.2GHz, 2MB L2 cache and a 800MHz bus, and a system memory capacity of

4GB DDR2 with 800MHz bus. Fixed point arithmetic simulation was performed using MATLAB's fixed point tool, further simulations were performed using C code complied using gcc version 4.4.3 on a fresh installation of Ubuntu 10.04 LTS. Ubuntu was installed as a guest in a virtual environment using VMWare Workstation ver. 7.1.2.

The target hardware platform is an Altera Cyclone II EP2C70F896C6 FPGA (Altera Website, 2011) device in the Altera DE2-70 development and education board (Terasic website, 2009). The EP2C70 FPGA is a low cost, high-volume, high performance FPGA for cost-sensitive applications. Table 1 below lists the main features of the EP2C70 FPGA.

Table 1 – Altera EP2C70 FPGA Features

Feature	***Count***
Logic Elements (LEs)	68,416
M4K RAM Blocks (4 kbits + 512 Parity Bits)	250
Embedded Memory (Kbits)	1,125
18-Bit × 18-Bit Embedded Multipliers	150
PLLs	4
Maximum User I/O	622
Differential Channels	262

All user hardware modules were developed in theVerilog language using behavioural modeling on the Altera Quartus II 9.1 Web edition IDE (Quartus II Website, 2011). Waveform analysis and system debugging was carried using SignalTap II logic analyzer which is bundled with the IDE. Memory debugging, and run-time loading and memory reading was carried through the Quartus In-System Content Memory Editor. The overall system was designed using Altera SOPC Builder 9.1. Processor control code was develped using NIOS II IDE 9.1 SP1 using the C language . Figure 7 shows the DE2-70 target board with main components highlighted.

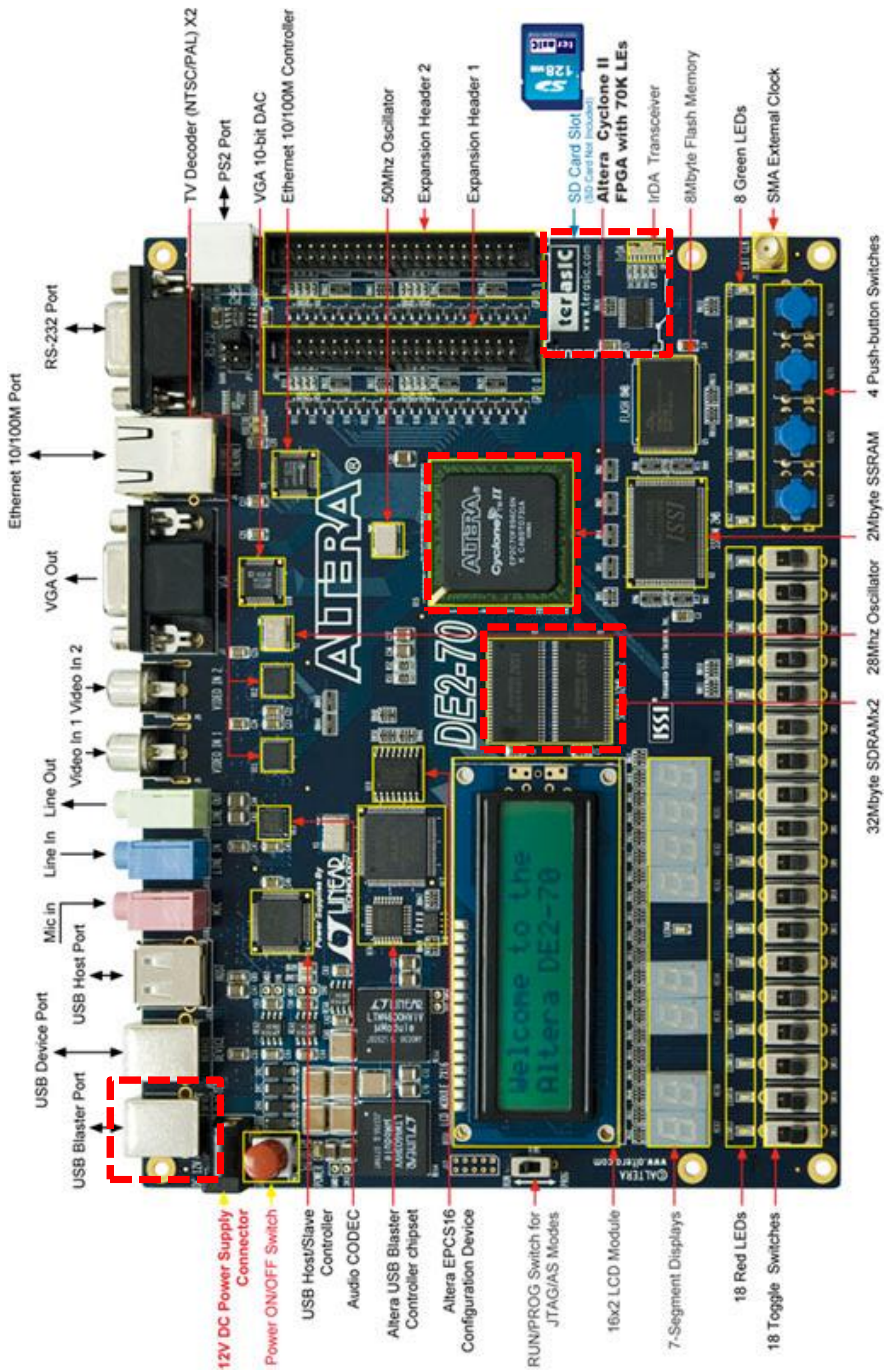


Figure 7 - Altera DE2-70 Development Board Layout. The components used are highlighted in dashed red

Chapter 3 – Algorithmic Analysis of Suggested Feature Extraction Techniques

In this chapter, we suggest four feature extraction algorithms for possible development on programmable hardware devices, namely, Density, 2D Forward Discrete Cosine Transform, Gradient Masks and Hu moments algorithms. We will briefly introduce each algorithm, the MATLAB simulation and sensitivity analysis results for each in regard to recognition rate accuracy and number of hidden layer nodes required (network architecture) as well as algorithm speed, we then propose for each a parallel hardware implementation, a thorough analysis and estimation of the hardware cost follows, finally, based upon algorithmic efficiency analysis, we conclude with suggesting the most feasible algorithm for implementation on hardware.

3.1. Suggested Algorithms

The main rationale behind choosing the following algorithms for feature extraction in our analysis was to find those algorithms which combine simplicity, good recognition accuracy, and room for parallelization. The latter was to satisfy our requirement of reducing recognition speed per word. Additionally, we preferred those algorithms whose computational core has been used one way or another in other application domains on FPGAs. Consequently, our choice settled on four algorithms, DCT, Density, Gradient Masks and Hu moments. The other feature extraction algorithms which we excluded from our investigation are based on structural features; loops, concavities, ascenders, descenders, intersections and word length. Even though these features are quite popular in offline handwritten recognition, they were abandoned. This is because the detection of such perpetual features is often unreliable on account of wide variation in handwriting style, thus

making their accurate detection quite a challenge (Madhvanath and Govindaraj, 2001). In addition, such algorithms further need a baseline estimation step which adds to the complexity of the system (Noaparast and Broumandnia, 2009). Aside from the complexity of their extraction, more than one structural feature is to be combined in one feature vector to enhance recognition accuracy (Khorsheed, 2002). On the other hand, DCT as an example gives good recognition results when used alone (Al-Khateeb et al, 2008). Also, the suggested algorithms are not restricted to holistic approaches but also used in segmentation-based methods at the character level, thus making our analysis more general (Khorsheed, 2002). The following subsections will introduce and discuss the implementation of the selected algorithms.

3.1.1. Two Dimensional Forward Discrete Cosine Transform (2D-DCT)

The discrete cosine transform expresses a signal, image or a function in terms of a sum of sinusoids with different frequencies and amplitudes, and is similar to the Fourier transform in that it transforms the signal or image from the spatial domain to the frequency domain. It helps separate the image into spectral sub-bands of differing importance. The DCT reduces redundancy and focuses the energy of the image in a very limited frequency range yielding a small number of features.

To elaborate, for an image A with width W and height H (in pixels), the corresponding output DCT is stored in image B with the same width and height. $A(x, y)$ represents the pixel value at row x and column y, where $A(x, y) \epsilon\ 0, 1$ which represent the pixel intensity based on the image format. $B(p, q)$ represents the corresponding DCT value at row p and column q, where $B(p, q) \epsilon\ R$ (the set of real values). The energy of DCT image is defined by:

$$E = \sum_{q=0}^{H-1} \sum_{p=0}^{W-1} B^2(p,q) \quad (7)$$

Most of the energy is focused in the DC component of the image that is *B (0, 0).*

For an $H \times W$ image, 2D-DCT can be expressed as:

$$B(p,q) = \alpha_p \alpha_q \sum_{x=0}^{H-1} \sum_{y=0}^{W-1} A(x,y) \cos\left(\frac{\pi(2x+1)p}{2H}\right) \cos\left(\frac{\pi(2y+1)q}{2W}\right), \begin{matrix} 0 \le p \le H-1 \\ 0 \le q \le W-1 \end{matrix}$$

$$\alpha_p = \begin{cases} \frac{1}{\sqrt{H}}, & p = 0 \\ \sqrt{\frac{2}{H}}, & 1 \le p \le H-1 \end{cases} \qquad \alpha_q = \begin{cases} \frac{1}{\sqrt{W}}, & q = 0 \\ \sqrt{\frac{2}{W}}, & 1 \le q \le W-1 \end{cases} \quad (8)$$

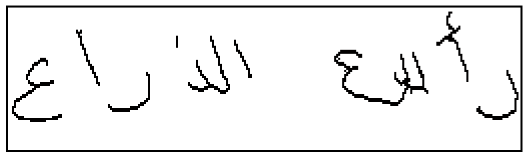
Another way of calculating 2D-DCT is the Row Column Decomposition (RCD) in which one dimensional DCT is applied on each row of the image, and then reapplied on each of the resulting columns. This method is widely used in many hardware IP cores, like those of CAST, VISENGI and BARCO-SILEX. The unitary DCT for a row '*a*' of width W is expressed as:

$$B(q) = \alpha_q \sum_{y=1}^{W} a(y) \cos\left(\frac{\pi(2y-1)(q-1)}{2W}\right) \quad q = 1,2, \dots W$$

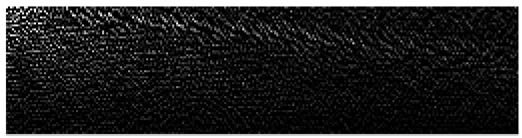
$$\alpha_q = \begin{cases} \frac{1}{\sqrt{W}}, & q = 0 \\ \sqrt{\frac{2}{HW}}, & 1 \le q \le W-1 \end{cases} \quad (9)$$

Where *b(q)* is the DCT value at location *q* in the output row *'b'*

The main feature of the DCT is that it reduces redundancy by concentrating the image energy in the upper left corner, Figure 8 illustrates the resultant DCT of the Arabic word Raas Al-Thra’a (رأس الذراع), notice the energy (white dots) concentration.



(a)



(b)

Figure 8 - DCT transfers the image from the spatial domain into frequency domain. (a) Original image in spatial domain (b) Image after DCT in frequency domain. Energy concentrates in the upper left corner (white dots)

Even though the DCT coefficients are computed for every pixel in the image with a total of 64x256 coefficients, it has been shown that the first 20 features preserve 99% of the image energy, and that selecting more features than 35 yields no improvement in recognition accuracy (Al-Khateeb et al, 2008). Based on this, a feature set of 36 DCT coefficients have been chosen in our tests, which considerably reduces the number of inputs to the neural network, and thus reducing the number of required multiplications in each neuron allowing for a feasible implementation. DCT features extraction is based on zigzag ordering, two orderings are usually used; Figure 9 illustrates the two methods, the technique in (a) was adopted.

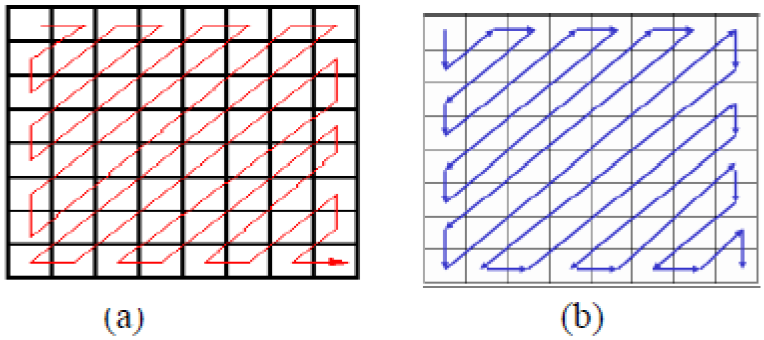


Figure 9 - The Two ZigZag DCT feature extraction methods

3.1.2. Density Features

The density technique is based on dividing the input word image into several $m \times n$ grids (windows or zones) in which the number of black pixels is summed and used as a feature. Different zoning or windowing strategies exist, some use fine windowing while others coarser windowing; in fine windowing, the size of the window is small, say 8x8 or 16x16, and thus yields a larger number of features. Coarse windowing uses larger window sizes and thus fewer features. Windows could be square, rectangular or any shape. Moreover, the chosen windows could interleave, or larger windows could be formed form a subset of smaller used ones.

In our tests, we have investigated three windowing strategies for the results of the possible window sizes and combinations are large. Yet we were mostly concerned that the window sizes be a multiple of 8 such that it would reduce system complexity in regard to memory alignment if this method was to be implemented on hardware. Initially 8x16 rectangular windowing scheme was chosen resulting in 128 feature vector, then a 16x16 windowing approach was investigated. Neither technique surpassed a 29% recognition rate using a neural network classifier for a range of hidden nodes from 6 - 100. The variety of Arabic handwriting styles could account for these poor results for it is highly unlikely that the

Arabic letters would fall in the same small zones for all writers. Therefore, a coarser windowing scheme was investigated, with four different-size window blocks, 32x32, 16x64, 32x64, and 32x128 resulting in a 44 feature vector. Figure 10 displays the density regions used and their order. This specific ordering is due to the use of the block processing function in Matlab “blkproc”. Since many blocks are subsets of larger blocks, the density of larger blocks can be calculated from smaller ones saving time.

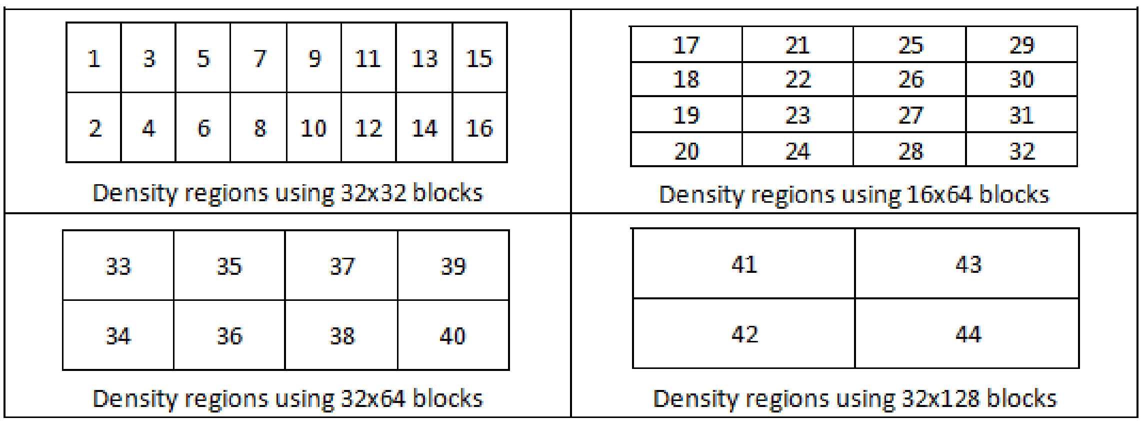


Figure 10 - The four feature sets used when coarse windowing is used in the density algorithm.

3.1.3. Gradient Masks

This feature extraction technique is adapted from the work of (Ebrahimpour et al, 2011) yet it was slightly modified such that it is simpler to implement in hardware. This technique is based on applying four scale invariant gradient masks to the image; these masks are shown in Figure 11 and correspond to scanning the image in vertical, horizontal and left and right diagonals using 3x3 masks, this would result in decomposing the input image word into four separate images where each only preserves pixels which match the mask.

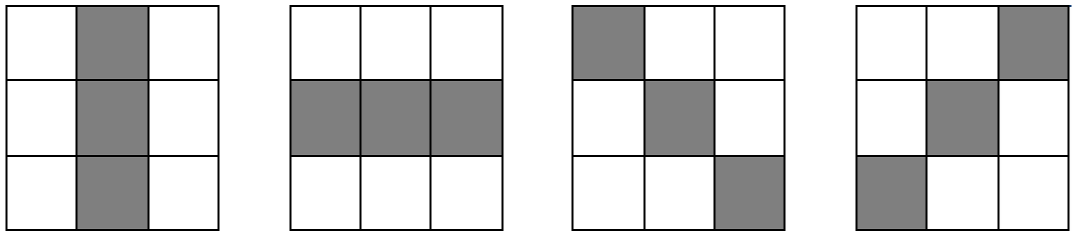


Figure 11 - The set of four gradient masks used to decompose the image

For each gradient mask, a same size zeroed image array is initialized, then as the mask scans the input image from left to right and from up to bottom by pixel width shifts, if a matching pattern occurs, the mask is copied into the new image at the corresponding scan locations.

To elaborate further, consider *A* and *B* as the input and output images respectively with same width and height, *A(x,y)* and *B(x,y)* ϵ 0, 1 correspond to the pixel value at row *x* and column *y* for both the input and output images, where *x* ,*y* are the row, column indices and *M* is the 3x3 mask. An example masking procedure for a vertical mask is thus expressed as:

```
If (A(x,y)==M(1,2)) && (A(x+1,j)==(M(2,2)) && (A(x+2,y)==M(3,2))
then
{
      B(x  , y) == A(x  , y)
      B(x+1, y) == A(x+1, y)
      B(x+2, y) == A(x+2, y)
}
```

The if statement condition could be rewritten as

```
      If (A(x,y) && A(x+1,y) && A(x+2,y)) == 1
```

where “1” is assumed to be the pixel value representing the colour black. This will reduce the number of arithmetic and logical operations from five to three.

The same procedure can be expanded above for all the remaining three masks by changing the indices to account for the mask shape. Figure 12 illustrates the results of applying all the four masks on the Arabic Word Tal Al-Ghuzlan (تل الغزلان):



Figure 12 - Four Extracted Images after Applying the Gradient Masks on the word Tal Al-Ghuzlan تل الغزلان

In the original technique, each of the extracted images was originally divided into eight regions around the image center as shown in Figure 13, and the number of black pixels in each region was counted and normalized to the total density of the word image, this provided eight features for each image producing a final feature vector of length 32. We have slightly modified the original image partitioning scheme which uses a triangular block around a centralized point of origin, and instead used a similar windowing scheme as described in section 3.1.2 above where the image was divided into eight 64x32 blocks. This would simplify the hardware implementation and reduce resources, this is largely because the number of image pixels needed to be stored at one time for each region is less, this would make it much more feasible to calculate features for the $m \times n$ regions from smaller regions similar to the approach adopted in the density algorithm. It is worth noting that no

normalization was considered contrary to the original algorithm. Recognition rates however, were quite close to the original algorithms despite our modifications.

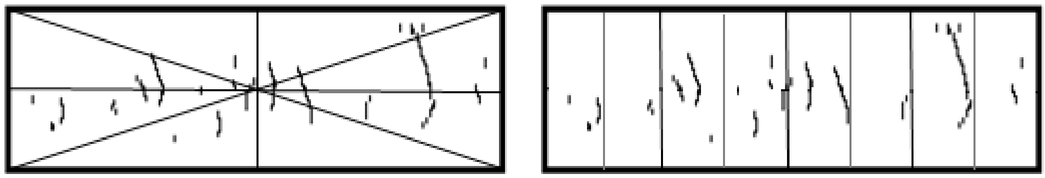


Figure 13 - Proposed density extraction regions by Ebrahimpour et all (Left). Our proposed Regions for feature Extraction (Right)

3.1.4. Hu Moments

Moments are image descriptors which can compactly represent certain properties of an image, for example, the first through forth order geometric moments can be interpreted as the area, variance, skewness and kurtosis of an image, thus providing a powerful statistical feature set for the identification and classification of images. Different moment types have been used in research for word and character recognition, centralized, Hu and Zernike moments are such examples. Hu moments, which are seven in total, are in fact derived from centralized moments up to order three, and they are invariant towards rotation, translation and scaling. Centralized moments are themselves derived from geometric moments. Hu moments are superior to centralized moments in that they are invariant to rotation and scaling whereas centralized moments are only invariant to translation.

The geometric moments M_{ij} for an $H \times W$ image are calculated as:

$$M_{ij} = \sum_{y=0}^{W-1} \sum_{x=0}^{H-1} x^i y^j \, A(x,y) \qquad (10)$$

Whereas centralized moments μ_{ij} are derived from the following equation:

$$\mu_{ij} = \sum_{y=0}^{W-1}\sum_{x=0}^{H-1}(x-\overline{x})^i(y-\overline{y})^j A(x,y) \tag{11}$$

Where $\overline{x} = \frac{m10}{m00}$ and $\overline{y} = \frac{m01}{m00}$

Another formula to calculate centralized moments up to order three is by using the following set of equations:

$$\begin{aligned}
\mu_{00} &= M_{00,} \\
\mu_{01} &= 0_{,} \\
\mu_{10} &= 0_{,} \\
\mu_{11} &= M_{11} - \bar{x}M_{01} = M_{11} - \bar{y}M_{10,} \\
\mu_{20} &= M_{20} - \bar{x}M_{10,} \\
\mu_{02} &= M_{02} - \bar{y}M_{01,} \\
\mu_{21} &= M_{21} - 2\bar{x}M_{11} - \bar{y}M_{20} + 2\bar{x}^2M_{01,} \\
\mu_{12} &= M_{12} - 2\bar{y}M_{11} - \bar{x}M_{02} + 2\bar{y}^2M_{10,} \\
\mu_{30} &= M_{30} - 3\bar{x}M_{20} + 2\bar{x}^2M_{10,} \\
\mu_{03} &= M_{03} - 3\bar{y}M_{02} + 2\bar{y}^2M_{01}
\end{aligned} \tag{12}$$

Hu moments are derived from scale invariant moments $\eta_{ij,}$ which themselves are derived from centralized moments following this equation:

$$\eta_{ij} = \frac{\mu_{ij}}{\mu_{00}^{1+\frac{i+j}{2}}} \tag{13}$$

The set of Hu moments can be expressed as follows:

$$\begin{aligned}
I_1 &= \eta_{20} + \eta_{02} \\
I_2 &= (\eta_{20} - \eta_{02})^2 + (2\eta_{11})^2 \\
I_3 &= (\eta_{30} - 3\eta_{12})^2 + (3\eta_{21} - \eta_{03})^2 \\
I_4 &= (\eta_{30} + \eta_{12})^2 + (\eta_{21} + \eta_{03})^2 \\
I_5 &= (\eta_{30} - 3\eta_{12})(\eta_{30} + \eta_{12})[(\eta_{30} + \eta_{12})^2 - 3(\eta_{21} + \eta_{03})^2] \\
&\quad + (3\eta_{21} - \eta_{03})(\eta_{21} + \eta_{03})[3(\eta_{30} + \eta_{12})^2 - (\eta_{21} + \eta_{03})^2]
\end{aligned}$$

$$I_6 = (\eta_{20} - \eta_{02})[(\eta_{30} + \eta_{12})^2 - (\eta_{21} + \eta_{03})^2] + 4\eta_{11}(\eta_{30} + \eta_{12})(\eta_{21} + \eta_{03})$$

$$I_7 = (3\eta_{21} - \eta_{03})(\eta_{30} + \eta_{12})[(\eta_{30} + \eta_{12})^2 - 3(\eta_{21} + \eta_{03})^2] - (\eta_{30} - 3\eta_{12})(\eta_{21} + \eta_{03})[3(\eta_{30} + \eta_{12})^2 - (\eta_{21} + \eta_{03})^2] \quad (14)$$

In this experiment, the image has been divided into eight rectangular regions of size 64×32 resulting in a total of eight bands, Hu moments were applied to each and the resulting feature vector of size 56 was used to train the neural network.

3.2. Algorithm Speed

All the proposed feature extraction algorithms have been coded in MATLAB, all but DCT were user coded and based on MATLAB's optimized built-in functions whenever possible. The 2D forward DCT was entirely based on MATLAB built-in function. Every algorithm was run over 100 times for a set of 8699 different images with total iterations of 869900. MATLAB's built-in "tic-toc" functions were used for time measurements. The time spent in reading the image and feature calculations was recorded and that spent for overhead as well, the net time spent executing the algorithm was averaged over all iterations and the results recorded as shown in Figure 14 below.

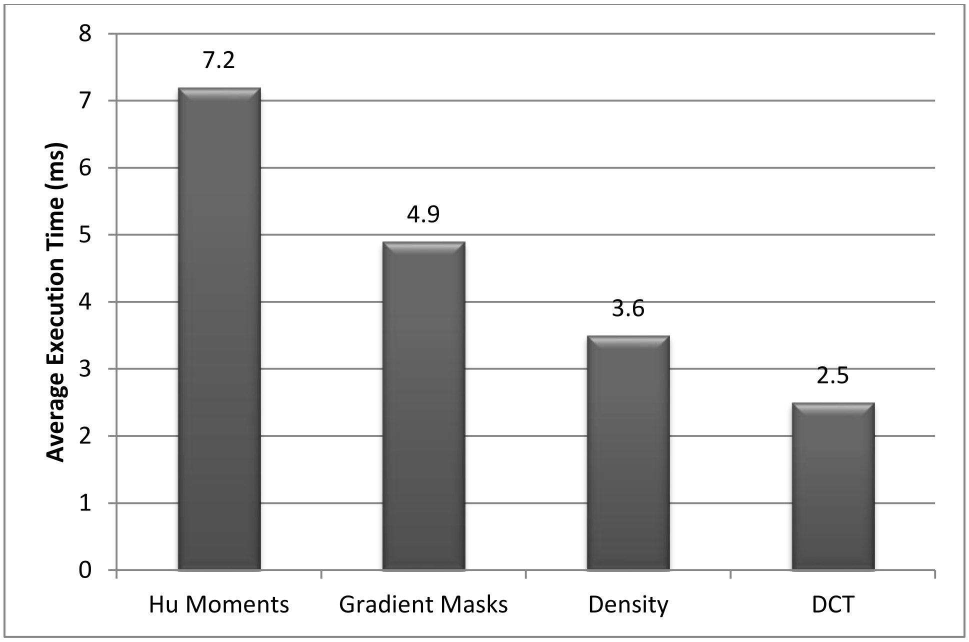


Figure 14 - Average execution time per algorithm based on extracting features from 64x256 pixels image over 869900 iterations.

3.3. Recognition Rate Sensitivity Analysis using Neural Networks

Since a lexicon size of 50 words was chosen for the reasons presented in section 1.3, we chose to select the images with the highest mode (where mode means the number of the word occurrences in the image set), for the more the samples, the better the training of the neural network will be. The images were chosen from sets A, B, and C from the IFN/ENIT database. It was found that if we set the mode of the images count to be at least 75, we will exactly have a set of 50 different images; the final set had a total of 8699 input images. For a complete listing of the words in the image set, the reader is referred to Appendix A. The set was used for tuning the neural network classifier and testing recognition rates. Default MATLAB neural network toolbox parameters were used to divide the set into 70% of samples for training, 15% for validation and 15% for testing. The indices of the test set was saved in order to retrieve the same test set for testing the hardware system. Every image was preprocessed by applying thinning and pre-scaling to a size 64×256 pixels. Initially,

ACDSee batch image resize tool was used to scale the images into the target size using the "Lanczos" filter, and then MATLAB's "bwmorph" function was used for thinning. The desired feature extraction method was applied to the image set and the resulting feature vectors were used as the inputs for the neural network for training, testing and validation.

The neural network is a feed forward back propagation neural network with three layers, input, hidden and output. One hidden layer node was only used for it is enough for the needs of pattern classification (Al-Khateeb, 2008). The output layer has 50 output nodes with each corresponding to an image word of the chosen data set. The number of hidden layer nodes varied from 6 to 100 in steps of two and recognition accuracy results were recorded for each architecture change. The *"tansig"* sigmoid function was used as the activation function for the hidden layer whereas the linear function *"purelin"* was considered for the output layer. Moreover, three network training functions were investigated, "*trainscg (scaled conjugate gradient method)", "traingdx (gradient descent momentum and an adaptive learning rate.)" and "traingda (gradient descent with adaptive learning rate)".* All training functions were used with their default settings. The simulations ran for a maximum of 50,000 epochs with a mean square error value set to 1.0e-6. Early stopping techniques through validation sets to avoid overfitting were used. The total number of features used as input to the first layer of the neural network is shown in Figure 15.

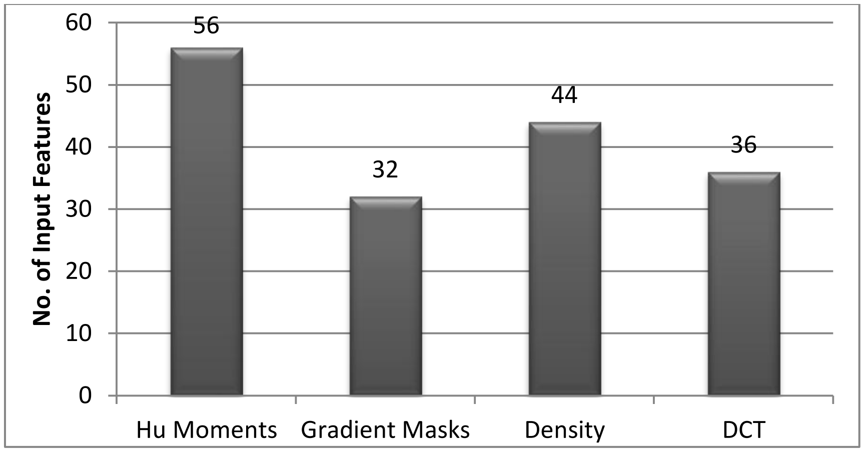


Figure 15 - Total Number of Input Features (Neurons) in the Neural Network for the four propsed algorithms

Figures 16 through Figure 19 illustrate the results obtained for all investigated methods. It is obvious that best results were obtained when the “trainscg” training function was used.

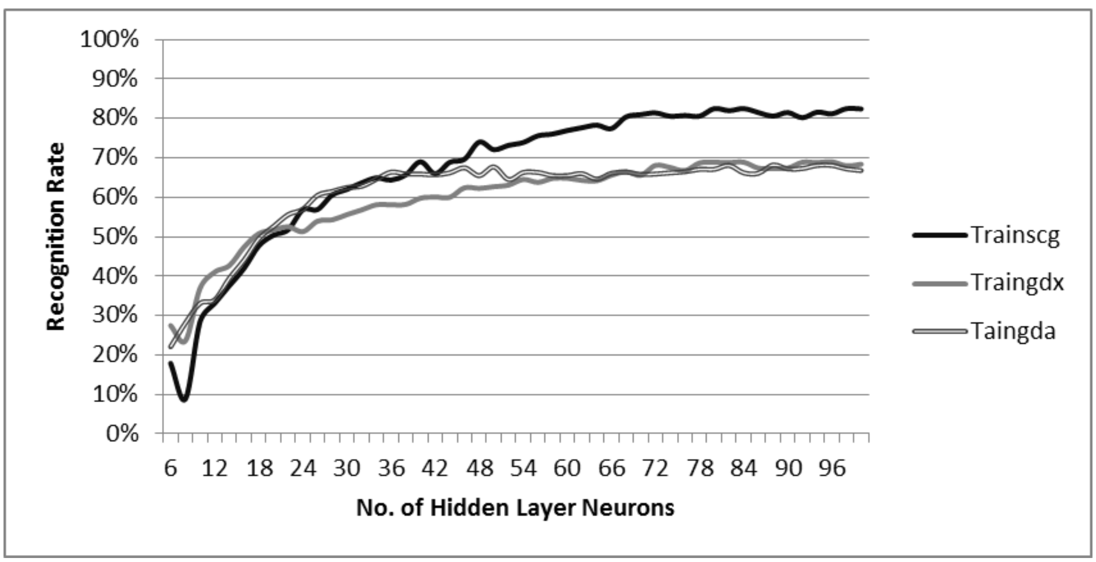


Figure 16 - Recognition Rate vs Number of Hidden Layer Neurons for Density Features using different Training Functions

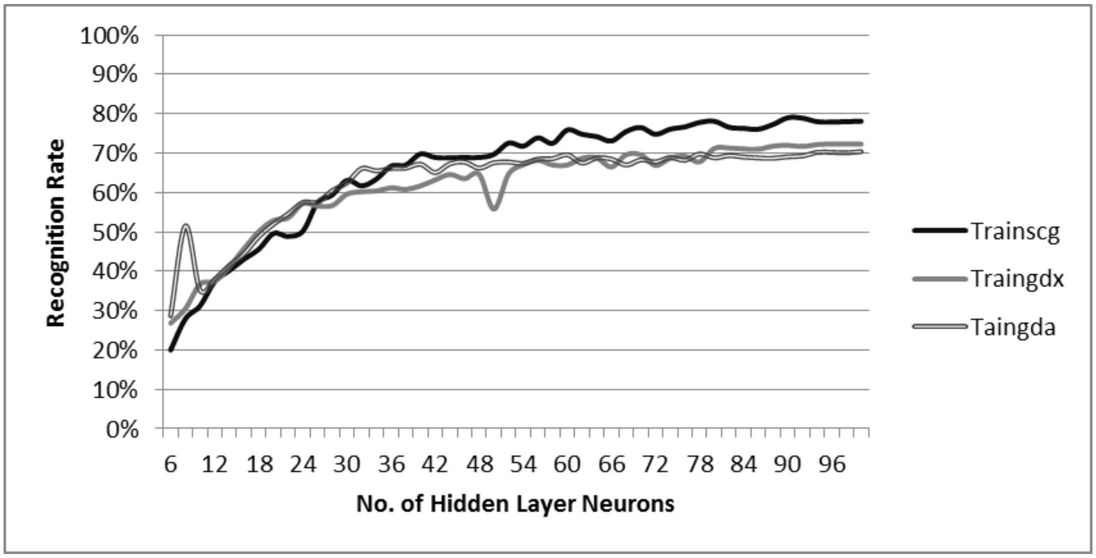


Figure 17 - Recognition Rate vs Number of Hidden Layer Neurons for Gradient Masks Features using different Training Functions

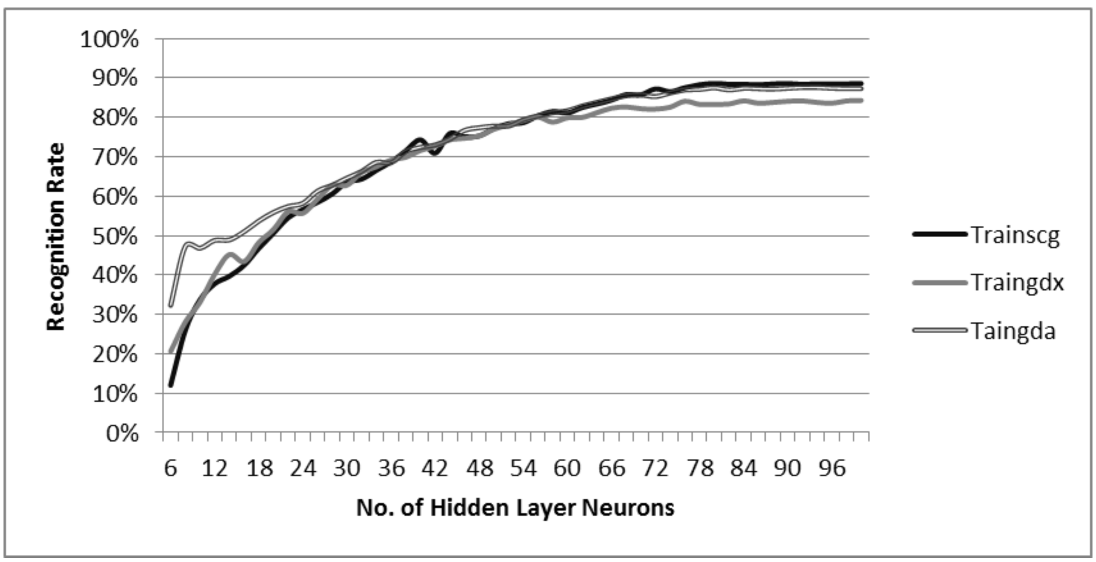


Figure 18 - Recognition Rate vs Number of Hidden Layer Neurons for DCT Features using different Training Functions

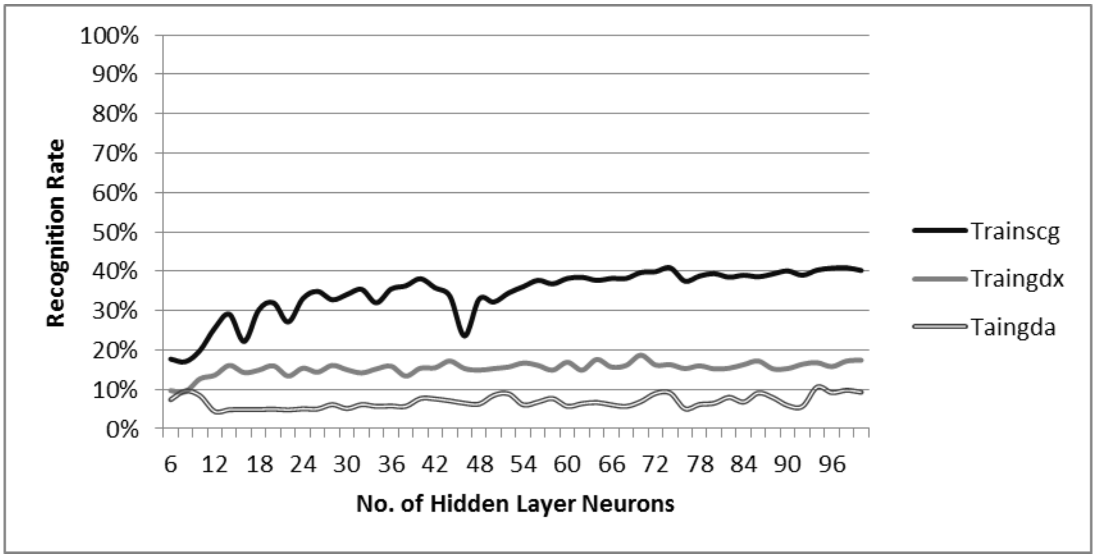


Figure 19 - Recognition Rate vs Number of Hidden Layer Neurons for Hu Moments as Features using different Training Functions

Figure 20 illustrates a comparison graph of the accuracy of all proposed feature extraction methods for a neural network structure with various number of hidden layer nodes. It is obvious that all techniques offer no improvement in recognition beyond 80 nodes in the hidden layer; DCT offers the highest rates yet in close proximity to Density and Gradient Masks. Hu moments fail to be of considerable value.

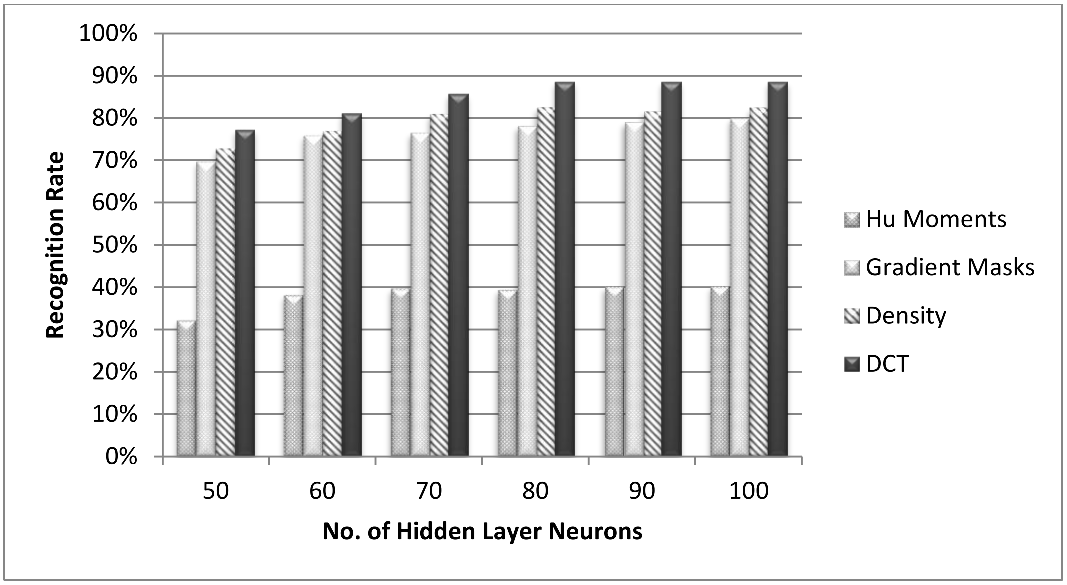


Figure 20 - Recognition Rates of Different Feature Extraction Techniques for selected Number of Hidden Neurons

Figure 21 shows the maximum absolute recognition accuracy for each of the techniques regardless of the number of the hidden neurons.

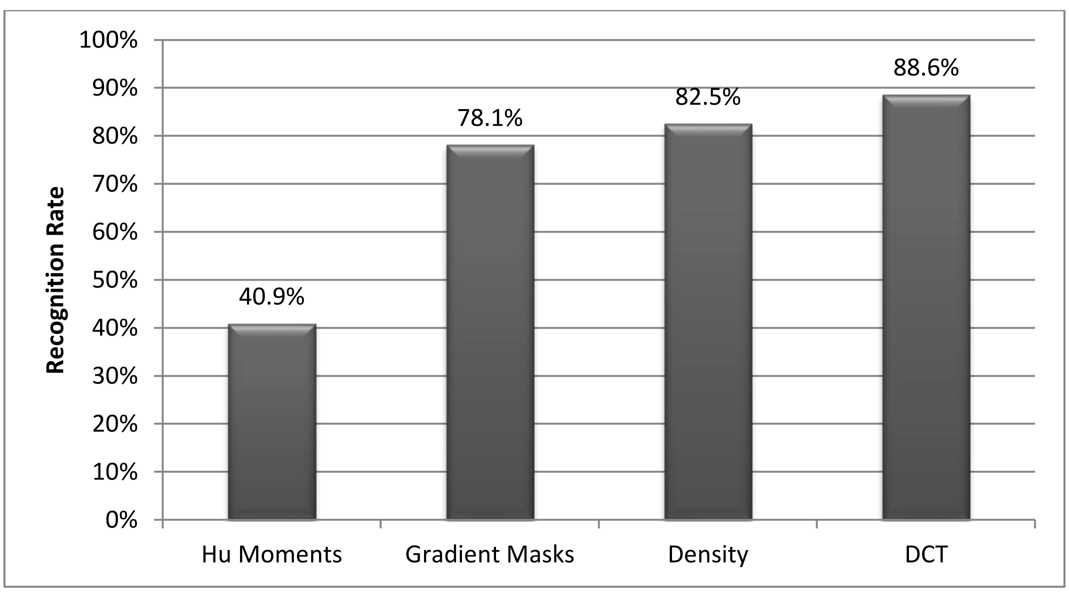


Figure 21 - Highest Recognition Rates Reported for the Proposed Feature Extraction Techniques

3.4. Algorithm Resource Usage Analysis and Estimation

In this section we attempt to estimate the hardware resources for each of the feature extraction techniques described in the previous sections. These estimations are based on the number of Logic Elements (LEs) used by each function as listed in Table 2. The figures in the table are exact and are obtained by building each function through the use of the Megafunction IP cores wizard in the Quartus II IDE. These costs are fixed for all the algorithms investigated.

Table 2 – The cost of arithmetic modules when implemented using the Altera's Megafunction IP cores on Cyclone II device (using balanced mode)

Function	***IP Core Name***	***Cost in Logic Elements (LEs)***
16x16 2D – DCT *(CAST)*	*2D Forward/Inverse DCT-CAST*	***1240***
16x16 multiply accumulate unit *(Altera)*	*ALTMULT_ACUM*	***400***
16x16 Integer multiplier *(Altera)*	*LPMMULT*	***336***
32x32 integer multiplier *(Altera)*	*LPMMULT*	***624***
M-bit Accumulator *(Altera)*	*ALTACCUMULATE*	***M***
M-bit integer adder/subtractor (No overflow) *(Altera)*	*LPM_ADD_SUB*	***M***
32x32 floating point multiplier *(Altera)*	*ALTFP_MUL*	***962***

*All above IP cores are free except for the CAST 2D DCT

All Altera arithmetic IP cores are instantiated through the MegaWizard Plug-In Manager in the Quartus II IDE, all follow nearly the same parameterization steps. The designer has the option to select the input data bus widths, while the output data bus width can be either automatically determined or manually set. The designer has the option to choose if some of the inputs are constant, or whether a multiplication function is a squaring function. One can

choose either signed or unsigned operations. Furthermore, the designer can choose whether the desired function be implemented using the logic resources or embedded hardware components/DSP blocks (i.e. embedded multipliers) if available. Finally, an optimization method could be selected, that is, whether the IP core will be compiled for area or speed optimization; a balanced optimization can also be chosen as in our case. For further details regarding the abovementioned IP cores, the reader is referred to (CAST 2D DCT Product Sheet, 2010) and (Altera Library of Parameterized Functions, Altera's website).

Furthermore, these estimations are only concerned with the computational core only and are directly derived from the actual algorithm, that is, no mathematical algorithmic optimization or modification has been proposed for this is out of the scope of this thesis. However, the proposed implementation is designed such that it runs in parallel on hardware to achieve speed advantages. In addition, all implementations are based on 16-bit fixed point arithmetic unless otherwise noted; this is due to the cost of the floating point units which is higher than their integer counterparts as shown in Table 2. It is assumed that the memory interface cost for every algorithm is the same, and it is further assumed that each module controller cost is far less than the computational core cost; therefore no cost estimation for controller and module interfacing is carried out.

3.4.1. Discrete Cosine Transform Hardware Cost Estimation

2D-DCT implementations and optimizations in either software or hardware have been targeting 8×8 and 16×16 block transforms with the former being the most prevalent; for 8x8 DCT is in extensive use in image processing and compression techniques. (Huang et al, 2009), (Tell et al, 2003) and (Shan, 2008). Consequently, rarely has larger size blocks been addressed in research, and due to this, no actual implementation exists for large size

2D-DCT to the knowledge of the author at the time of writing, thus no reference implementation or optimization concerning large $H \times W$ DCT transforms can be used. Nonetheless, it is worth noting that IP cores such as that offered by CAST for an up to 16×16 2D forward DCT take up about 1240 LEs with 9 embedded multipliers on our target device: the Cyclone II FPGA, it is highly expected that due to that our proposed DCT algorithm runs on the whole image of size 64×256, that the cost will be significantly larger. On the other hand, since our input image is in binary format, the DCT problem will be quite simpler to implement yet still resource exhaustive.

We will estimate the resource usage of the 64×256 DCT based loosely upon direct algorithm implementation and extending and modifying the design presented by (Rosenthal, 2006) for an 8×8 DCT. Though the base design applies the 2D-DCT equation directly, the approach we propose herein is based on Row Column Decomposition (RCD), that is to perform 1D DCT on the rows, then apply it again on the decomposed rows on a column basis. Moreover, all cosine values are pre-computed and stored in memory (ROM) such that the only unknown value is the binary pixel value; this will reduce the complexity of the computational core reducing the LE utilization on expense of higher memory usage for the cosine look up tables. We opted for applying the 1D DCT transform in two stages rather than direct 2D-DCT for FPGA implementation because it reduces the amount of space the cosine multiplication lookup table occupies in memory, extending the technique in (Rosenthal, 2006) requires 256×256×64×64×2 bytes (512 MBytes) accounting for all multiplication possibilities whereas using 1D transform reduces the look up table size to (256×256 + 64×64) × 2 = 136 Kbytes, a 3 order magnitude reduction factor! Yet still, a 136 Kbytes closely fills the target development board internal memory.

Given that the input image has binary input pixel values, the 1D row DCT simply reduces to summing the cosine values whenever the input pixel has the value of one and the final value of the summation is multiplied by a constant.

To achieve higher performance, parallel computation is considered at the row level, that is, referring to equation 9, *b(q)* for all q = 1 … 256 are computed in parallel. For each *b(q)*, a 16 bit accumulator is proposed and the final summation value is multiplied by the constant α_q. 16-bit accumulator is used for all the summed values are in 16 bit fixed point format with one sign bit and 9 fraction bits and it has been empirically shown that no intermediate or final summation values exceed the designated integer span of -64 to 64. The cost of a 16 bit accumulator as shown in Table 1 is 16 LEs while a 16x16 integer multiplier costs 336 LEs entailing a total cost of 256×16 + 336 = 4432 LEs per row.

In column DCT however, the new input values for *b(q)* are fixed point instead of binary, consequently the accumulator will be replaced by a multiply-accumulate unit which costs 400 LEs per unit followed again by a 16 bit integer multiplier . Therefore for 64 bit long column, and parallel computation for each bit, a total of 64×400 + 336 = 25936 LEs is required. Therefore, the final estimated cost of the computation core is 30368 LEs. Table 3 summarizes the costs for the proposed implementation.

Table 3 – Estimated Total Cost of the DCT Parallel Implementation for a 64×256 image

	Type of Unit	*Cost of Unit*	*Number of Units Needed*	*Total Cost*
Row DCT	16-bit Accumulator	16	256	4096
Row DCT	16×16 Multiplier	336	1	336
Column DCT	16×16 Accumulator Multiplier	400	64	25600
Column DCT	16×Multiplier	336	1	336
			Total Cost	**30368**

If maximum throughput is to be considered, all rows then columns need be computed in parallel. Extending our discussion above to account for all rows and columns being computed in parallel, a total cost of 6,923,264 LEs will be required. This figure surpasses the total number of LEs of today's state of the art high density FPGAs which only provide up to 1.19 million LEs (Altera, 2011). Therefore, the amount of parallelism and DCT acceleration and performance needed is limited by the high integration costs. Eventually, the optimum number of rows and columns to which DCT is applied in parallel is a design parameter which need be analyzed. This is out of the scope of this thesis.

3.4.2. Density Hardware Cost Estimation

Estimating the cost of the Density technique is by far the easiest and simplest, we start by estimating the cost of adding the elements in the 32×32 block which is the chosen basic block, it has been found that the cost of adding 32 single bit elements in a single cycle requires 64 LEs (no accumulator used), and to take advantage of row independence in each block, 32 rows are processed in parallel for a total estimated cost of 2048 LEs, the total cost of summing all resulting row summations together takes 274 LEs. Actually, to follow the proposed algorithm in section 3.1.2, the total sum in the 32×32 block is carried out by adding two intermediate summations, those of the upper and lower 16×32 blocks of the 32×32 block, this would make it easier to derive the second set of features whose block size is 16×64 for the intermediate sums of the upper group block is retained for next features set calculation. To illustrate, and referring to Figure 10, feature No. 17 of size 16×64 will be obtained by adding the intermediate upper half sums of blocks 1 and 3, and feature 18 will be the result of subtracting the upper half sums of the two square blocks 1 and 3 from feature 17.

Assuming the worst case that the maximum sum is 1024 for a block – which is in reality hard due to preprocessing, specifically thinning – the cost of 10 bit addition/subtraction is 10 LEs. That is for subsequent feature derivation 10 LEs are used when adding two subfeatures, and 20 LEs if a further subtraction operation is needed.

Consequently, the total cost of obtaining the second feature group is 10×8 + 20×8 = 240, while obtaining the third group requires only 80 LEs, Finally, the last feature group costs 40×4 = 160 LEs. All the previous feature set extractions are computed in parallel.

Thus, the total estimated cost is 2048 + 274 + 240 + 80 + 160 = 2802 LEs

3.4.3. Gradient Masks Technique Cost Estimation

It will be assumed that this technique will be implemented as follows, the original image will be loaded into a 3 line × 256 bit internal register buffer which is being updated by feeding it the new line of the input image, the size of three is chosen to match that of the mask size, each mask comparison consumes 1 LE, to fully parallelize the operation, the mask comparison will be done all in parallel requiring a total of 256 LEs per mask. The 4 masks will be all calculated in parallel such that the original image will only be loaded once into the buffer. Consequently, a total of 1024 LEs is estimated for all masks. The buffer itself is implemented in the M4K memory blocks internal to the FPGA not utilizing any logic resources.

After the images are extracted, an implementation is similar to that of the density is suggested, that is compute parallel rows in blocks and derive other larger blocks from smaller ones, assuming features are extracted from each new image in sequence, the expected resource usage here will be 2048 + 274 + 80 = 2402 LEs assuming the 64×32 are

to be computed from smaller 32×32 blocks (2048 for row calculation, 274 for block rows summation and 80 for summing two 32×32 block), if the 64×32 block is considered to be a whole unit instead of two consequent 32×32 blocks, then the resources will be approximately double that figure. Assuming the former, the total estimated number of logic resources is 2402 + 1024 = 3426 LEs and 4 × 256×64 + 3×256 = 66304 bits (~8KB) of memory for holding the 4 images and the buffer.

3.4.4. Hu Moments Hardware Cost Estimation

Moments in general are remarkably known for their high computational complexity, yet this complexity can be drastically reduced given that the input images have binary pixel values. Given the general equation:

$$M_{ij} = \sum_{y=0}^{W-1} \sum_{x=0}^{H-1} x^i y^j \, A(x, y) \tag{15}$$

And since the values of $A(x, y)$ are binary, and that x and y are the pixel coordinates which are previously known, the moment calculation problem can be reduced to a simple conditional summation one. (Paschalakis et al, 2003) proposed a system for parallel geometric moment computation on FPGAs based on parallel accumulator architecture, we will base our implementation on a similar idea for the implementation of the former was not fully disclosed. For all pixels in the *64×32* block, one can save the costs of the multipliers and raising powers by pre-computing *the* x^i y^j for all values of x and y for all required moments and store them in FPGA memory. For each row in the 64×32 block, the associated pre-computed values can be retrieved and the values summed using accumulators if the $A(x,y)$ == 1, where “1” is assumed to be the pixel value of black in the

image format. For each moment, and for each row, all accumulators work in parallel. Figure 22 below illustrates the proposed system implementation.

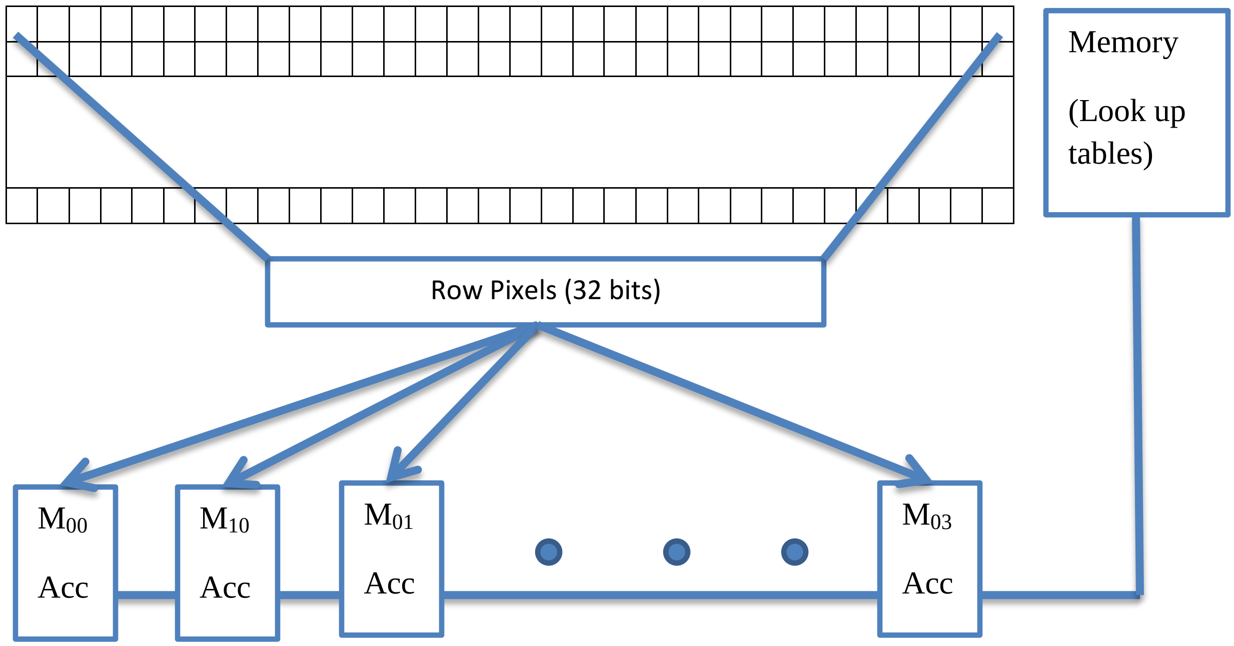


Figure 22 - Proposed parallel hardware implementation for calculating generic moments

Estimating memory cost is straightforward: in a block we have 64×32 = 2048 values to be pre-computed for each moment- that is the values of $x^i \times y^j$ - for a total of 2048 × 6 = 12288 values for all six moments (M_{00} need no tables for $x^i \times y^j$ always equals 1). Since the largest input value size is attained when calculating M_{30} for all bits of the last row of a block, which is equal to 63^3 = 250047, each value will take 4 bytes in size for a total memory size of 48 Kbytes (a size of 4 bytes was clearly chosen for memory alignment constraints).

We show below in our worst case analysis, that 4 bytes is enough for all intermediate values in consequent operations. The worst case analysis assumes the impossible input where there is no word image but pure black background, that is for all values of A(x, y) are the same and equal 1.

In Table 4 below, we estimate the cost for each accumulator shown in Figure 21 for the seven required moments.

Table 4 – Hardware Cost Estimation of Extracting the General Moments

M_{ij}	Maximum Sum in a Row	Accumulator cost (LE) per Row	Total Accumulator Cost (LE) for all Rows	Max Summation in a Block	Block Accumulator Cost (LE)	Total Est. Cost per Moment (LE)
M_{00}	31	5	320	1023	10	***330***
M_{10}	2016	11	704	64512	16	***720***
M_{01}	496	9	576	31744	15	***591***
M_{20}	127008	17	1088	2731008	22	***1110***
M_{02}	10416	14	896	666624	20	***916***
M_{30}	8001504	23	1472	130056192	27	***1499***
M_{03}	246016	18	1152	15745024	24	***1176***
M_{21}	1968624	21	1344	42330624	23	***1367***
M_{12}	656208	20	1280	20998656	22	***1302***
					Total	***9011***

Following is a brief description of the notations found in the table:

- *Maximum Sum in a Row:* the maximum possible summation to be ever calculated in any row in the block, derived from the last row of the block for it always has the largest sum due to its high indices order. Used to estimate maximum accumulator width.

  e.g. For M_{00}, max sum is that of last row where $x^i \times y^j = 63 + 63 + 63 + \dots 63$ (32 times)

  Similarly, for M_{21}, $63^2 \times 0 + 63^2 \times 1 + 63^2 \times 2 + .. + 63^2 \times 31$

- *Accumulator cost (LE) per Row:* it has been shown in Table 1 that an accumulator of bit width M requires M logic elements. The cost is thus based on the number of bits needed to encode the maximum sum value retrieved in the *Maximum Sum in a Row*

- *Total Accumulator Cost (LE) for all Rows,* to simplify estimation, we assume that for all rows, same size accumulators of that of the one used in the row of maximum sum are used. Consequently, the total accumulator cost for block = Maximum Row Accumulator size × No. of rows (64)
- *Max Summation in a Block:* assuming *A(x,y)* is one for all the block indices, all $x^i y^j$ are to be summed, this is used to estimate the width of the accumulator used to sum all the resulting summations of all rows in the block.
- *Block Accumulator Cost (LE):* similar to *Accumulator cost (LE) per Row* field, the number of bits to encode the largest block sum is used as the total width of the final accumulator.
- *Total Est. Cost per Moment (LE)* equals the summation of Total Accumulator Cost (LE) for all Rows and Block Accumulator Cost (LE) fields

In order to calculate $\bar{x}$ *and* $\bar{y}$, two dividers are needed each costing 265 LEs each. The result is 16 bits wide based on M_{00}, M_{10} and M_{01} sizes.

Equation set (12) is to be used in calculating the central moments, we assume now that the inputs are in 32 fixed point formats, converting the input values to this format is ignored in the estimation. Now, a 32×32 multiplier costs 624 LEs and a 32-bit subtractor or adder costs 32 LEs. Based on these figures, fully parallelizing the calculation of the central, scale invariant moments η and Hu moments is quite resource demanding and doesn't fit in the target FPGA. Thus, we will use 5 multipliers and 3 adder/subtractor to obtain the μ moments, and that these components will be reused for the Hu moments extraction suggesting a total cost of 3216 LEs. As for the real powers in η calculation, we will assume an implementation where the values are rounded to their nearest integer, (Bhowmik et al,

2006) have experimentally observed that this estimation produced good results for the purpose of moment calculation in hardware.

In the end, based on the suggested parallel implementation above and the simplifying assumption made, the total estimated cost for the moments extractor is 9011 + 265 + 265 + 3216 = 12757 LEs

3.5. Efficiency Analysis and Implementation Recommendations

In previous sections, recognition rates, speeds of serial MATLAB implementation as well as estimated hardware costs for the four suggested feature algorithms have been discussed and analyzed. In this section, we summarize the previous results, analyze them and suggest which algorithm is best suitable for implementation in hardware. Figure 23 summarizes the estimated costs for the algorithms in terms of Logic Element resources, while Figure 24 presents a similar cost analysis but based on the total memory space consumed.

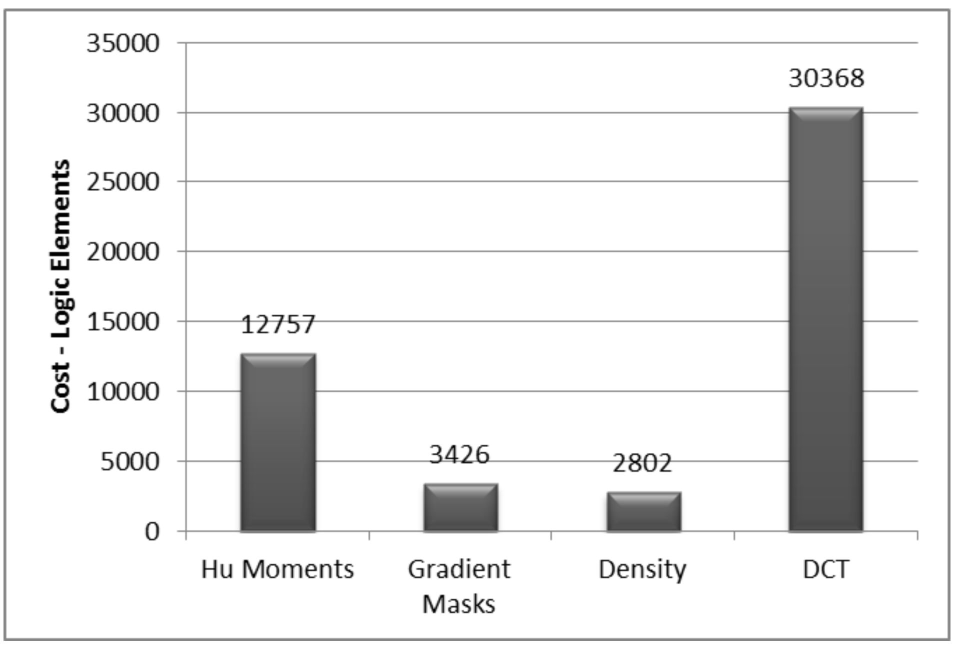


Figure 23 - Estimated Number of Logic Elements of the Cyclone II FPGA to be used in implementing the Computational Core of the proposed algorithms based on the analytical analysis of chosen hardware implementation.

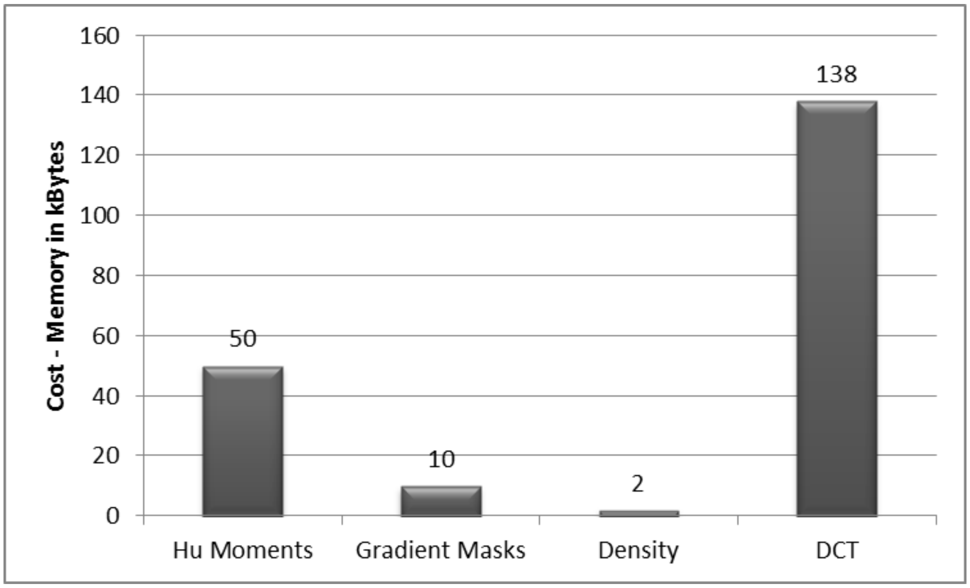


Figure 24 - Estimated Internal FPGA Memory Resources in KB to be used in implementing the Computational Core of the proposed algorithms based on the analytical analysis of chosen hardware implementation. The 2KB image size is included in the figures.

To simplify analysis, we will normalize and combine the figures above. Unfortunately no reliable reference was found to properly dimension the memory cost to the logic elements cost for Altera has discontinued the practice of announcing equivalent gate measures (Altera website, 2006), therefore, no precise and accurate information on the number of transistors each has or the area they occupy in the Cyclone II FPGA to properly scale and compare total cost. However, according to Leventis et al (2005), in the Cyclone EP1C20 device, the logic element fabric had five times as much area as that of the memory block area where the device has around 20,060 LEs and 294,912 bits of memory. Given that Cyclone and Cyclone II devices have nearly similar LE architecture (Altera Cyclone and Cyclone II handbooks), and since our device has 68,416 LEs and 1,152,000 bits of memory, and drawing on the LE to memory area ration in the Cyclone EP1C20, the assumption undertaken herein was that the LE area cost is ~5 times higher as that of memory. Figure 25 illustrates the normalized estimated hardware costs for all proposed

feature extraction methods in terms of logic and memory resources, the total normalized estimated hardware cost after adjusting the memory resource cost is also illustrated.

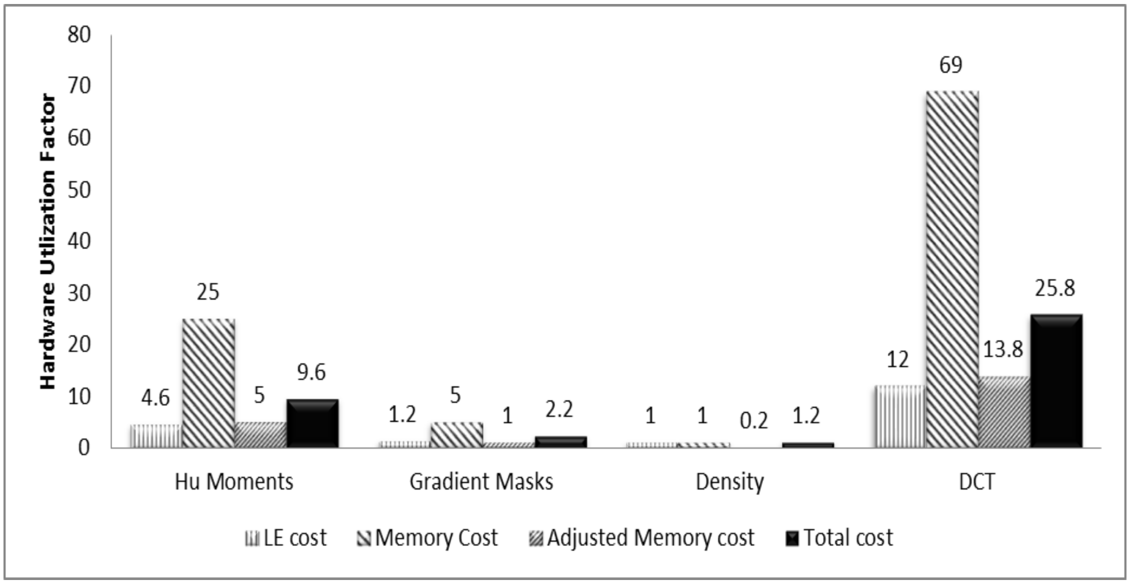


Figure 25 - Hardware Utilization Cost Factor for all proposed feature extraction techniques normalized to the Density Technique. Memory cost is adjusted by a factor of 1/5 to in order to properly estimate the total hardware cost.

Figure 26 illustrates the efficiency of the proposed techniques for feasible migration onto FPGA hardware. The efficiency was derived by dividing the maximum recognition rate of the proposed technique over the normalized total cost. It is clear that the density is superior to the other techniques and therefore our FPGA implementation for the OCR system will be based on extracting density features.

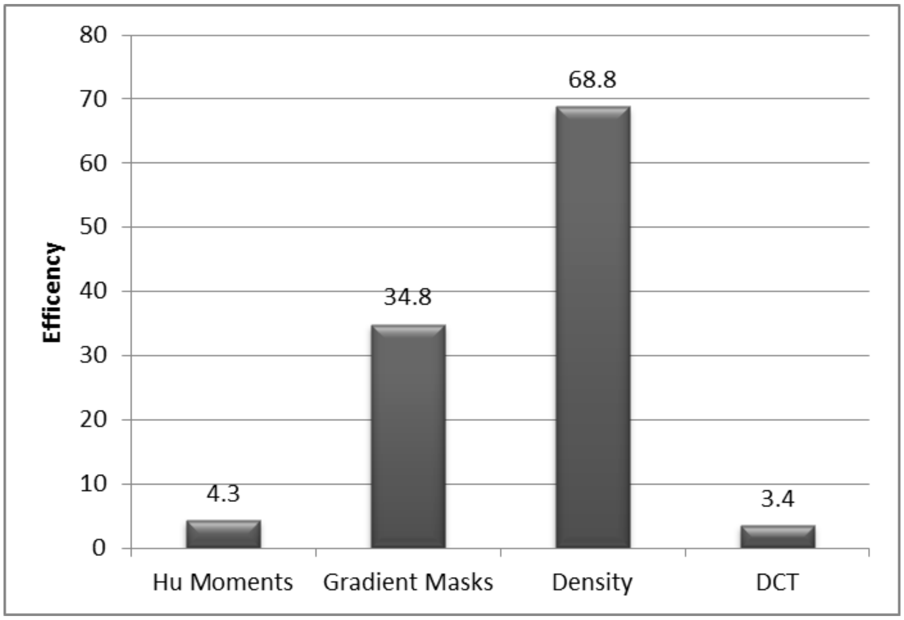


Figure 26 - Accuracy / Cost Efficiency Bars

Chapter 4 - Hardware Implementation

In this chapter we present the general hardware design framework where we introduce the main system components, system interconnections, and module communication. We further elaborate on the flow of operations of the general system. Finally the design of the OCR system modules is discussed with detailed flow of operations presented for both of the feature extraction as well as the neural network recognition based engine. Design options and justifications are introduced whenever necessary. The overall system architecture was designed using Altera's SOPC builder while the OCR modules were implemented in Verilog using behavioral modeling.

4.1. Hardware Design Framework

In this design we sought an implementation in which the role of the PC is minimum aside from device programming. It is only restricted to displaying intermediate and final stage results for debugging purposes on the IDE terminal window, all required operations from acquiring the images in the test set, loading them into the FPGA, the two phases of the OCR system, as well calculating overall recognition rate is implemented on FPGA.

A software hardware co-design approach was considered in our implementation to meet the timing constraints of the thesis and simplify design options; a NIOS II processor core was considered which is responsible for managing image transfer, reading back the outputs of the neural network, and calculating the final recognition rate of the test set. Its role was extended to be the main controller of the system, that is issue start signals for the next phases and monitor their phase end signals for appropriate system management.

A design in which the test image set is stored on an external SD-CARD was preferred over transferring them through serial communication from the PC. This is because the SD-CARD drivers are readily available from Terasic (the DE2-70 Board manufacturer) and that Altera offers the option to access files on the PC in debug mode only which is not a convenient solution (Altera, 2011 "Developing Programs Using the Hardware Abstraction Layer"). A system timer module is required for system drivers operation and thus included in the design. Figure 27 outlines the main system components.

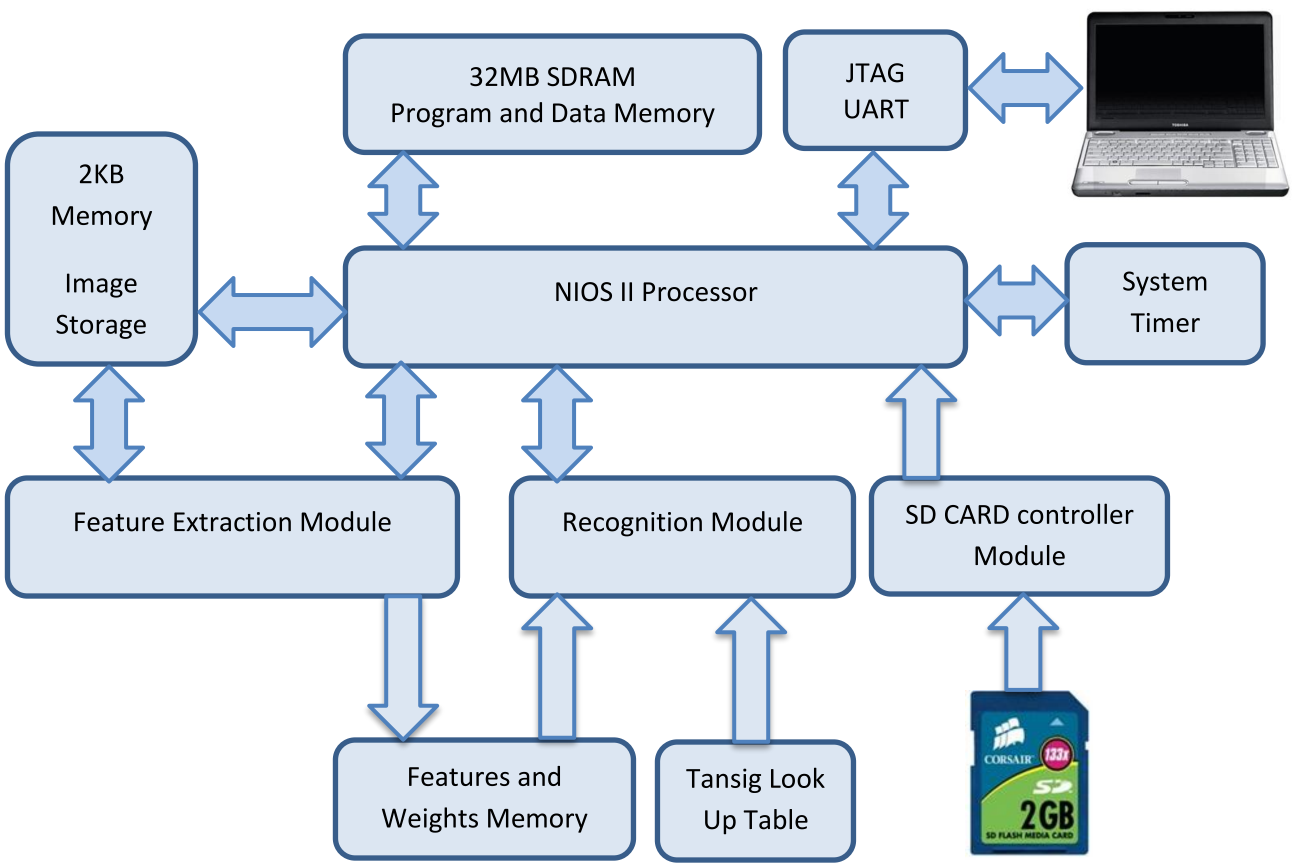


Figure 27 -Overview of system components

The JTAG UART is used to configure the FPGA for the requested system design above, download the compiled C user codes and libraries of the NIOS II processor to the SDRAM, as well as serve as the communication interface between the hardware design and the IDE

terminal for monitoring and debugging purposes at all levels of the design. The system starts by searching for and mounting the SD CARD, reading the FAT tables and loading the first image to the internal memory blocks, since the images are all binary, a total of 2Kbytes of data are transferred for fixed image sizes of 64×256 bits. The NIOS II processor signals the feature extraction module to start by setting the start bit inside the module control register, this register is internally checked at every clock cycle and reset by the module when the system finishes the task at hand. The resulting feature vector is stored in the "Features and Weights" memory. When the start bit of the feature extraction module is polled as zero, which means that the module has finished its designated task, the NIOS II processor enables the recognition module; the final output vector of the recognition module is internally stored in the module and is read by the NIOS II processor. The database documentation demands that recognition is done by returning the associated zip code of the Tunisian city/town name rather than the word itself (IFN/ENIT database Readme file, 2003) therefore the neural network output vector is used to look-up a zip code. The retrieved zip code is compared against the actual zip code of the image – to simplify the process; the four initial characters of the image name correspond to the actual zip code. Records are kept for the total number of words read and those correctly recognized from which recognition accuracy is obtained. Figure 28 describes the abstract high level operation flow of the system except for the feature extraction and recognition modules which are outlined separately in subsequent sections.

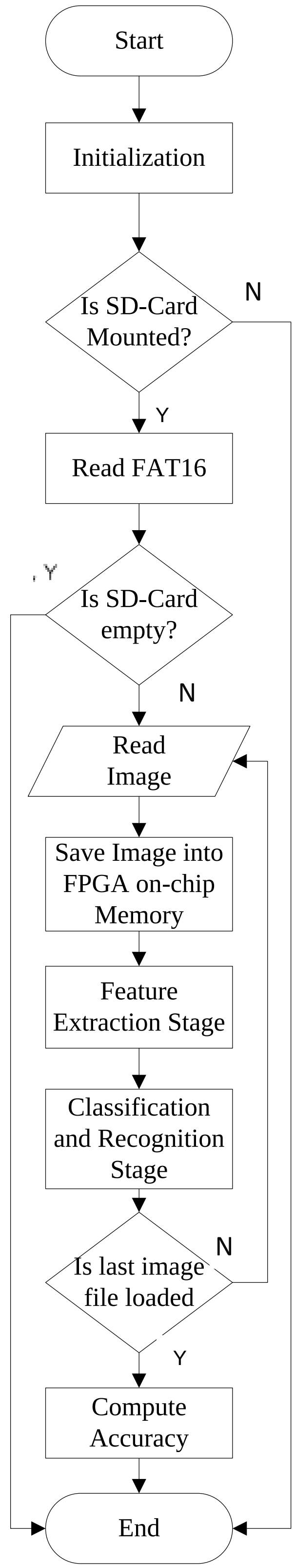


Figure 28 - General system Flow Diagram

4.2. NIOS II Processor Based System and Custom-Module Communication

A NIOS II processor system is equivalent to a microcontroller or “computer on a chip” that includes a configurable soft-core processor, a combination of peripherals and memory (either on-chip memory or interfaces to off chip memory) on a single chip. The NIOS II processor comes in three versions, fast (NIOS II/f), standard (NIOS II/s), and economical (NIOS II/e); the former is the fastest with up to 218 Dhrystone MIPS when it runs at a maximum frequency of 185MHz. It further adds advanced features like dynamic branch prediction, hardware divide and multiply operations, as well as instruction and data caches. Despite consuming more logic resources (around ~2.5X higher than the economical version), it was adopted in our design for faster image loading and overall operation. In our design, the processor runs at a clock frequency of 100MHz. The economical version is the only free version of the NIOS II family though.

4.2.1. Standard Peripherals

Altera provides a set of peripherals commonly used in microcontrollers, such as timers, serial communication interfaces, general-purpose I/O, SDRAM controllers, and other memory interfaces for immediate use in system designs. These peripherals can be readily instantiated in the System on Programmable Chip (SOPC) builder tool. Each of these peripherals has a set of control registers through which the NIOS II system manages the operation of the peripheral, Hardware Abstraction Layer (HAL) drivers are provided to facilitate reading and writing to these registers by hiding complexity and implementation details. Figure 29 shows the Arabic OCR NIOS II based system with all component interfaces, base addresses and interrupts.

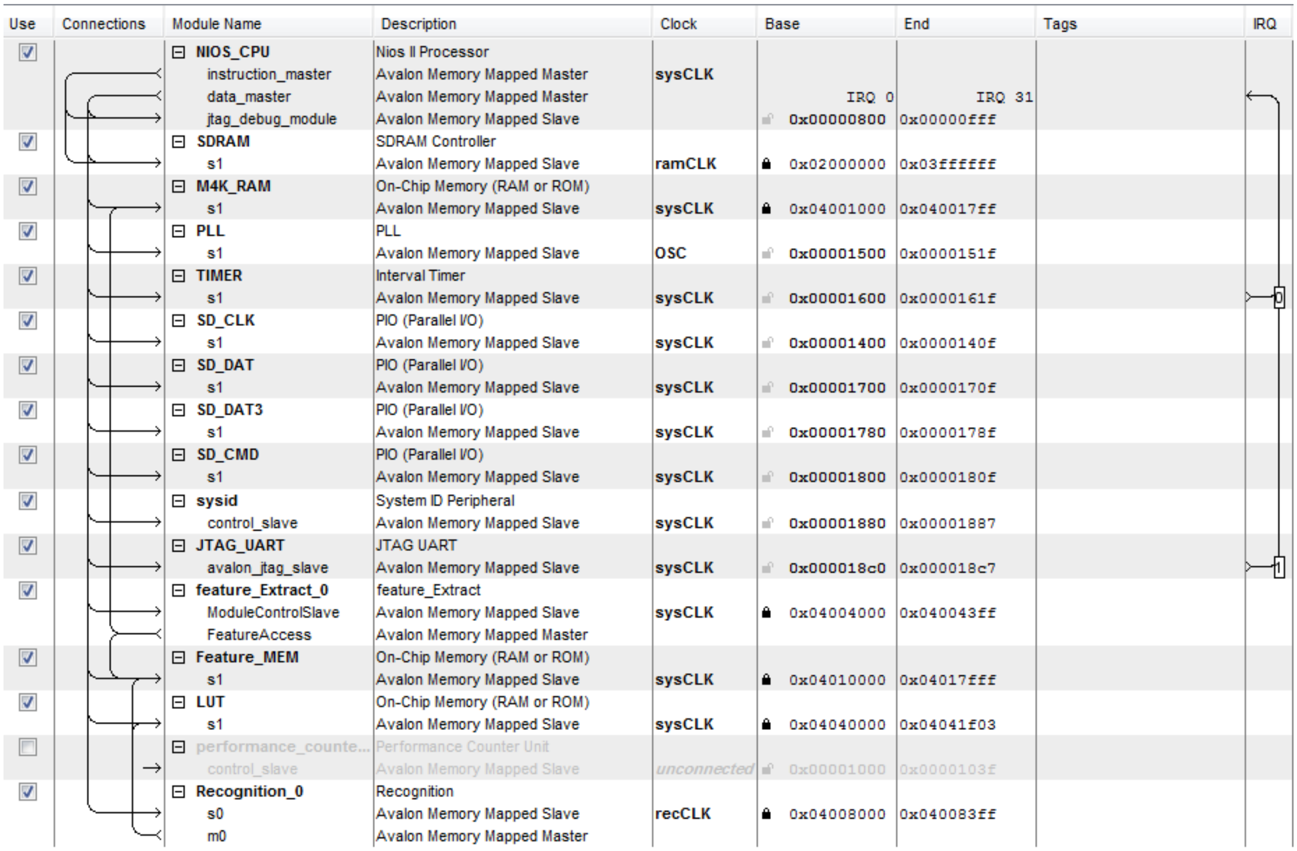


Figure 29 – System design in SOPC builder window. System modules, interfaces, clock sources, base address(es) and interrupt vectors are shown. The base address(es) represent(s) the address which serves as the reference point to read/write the internal registers in slave modules, or memory locations in a memory block.

This system is based on a NIOS II/f processor as the system's core, the processor was configured with default settings of 4Kbytes instruction cache, 2Kbytes data cache with 32Bytes data cache lines. JTAG level 1 debugging was used. SDRAM is the main system memory, the processor reset and exception vectors are changed such that they point to addresses in SDRAM at their designated default offsets. The default SDRAM profile and timing is used as provided by the SOPC builder. A phased looked loop is used to feed system components with their different clocks, the NIOS II processor, system peripherals and the feature extraction module run on 100MHz (sysCLK), the SDRAM clock is at the same system frequency albeit having a lag of -3ns as required in Altera's datasheets. The

recognition clock is set at F_{MAX} of 45MHz, the maximum achieved for the recognition module. A JTAG UART module is automatically added to allow for system interconnection with the PC for program download and debugging purposes. A timer is also added to the system as required by system drivers, timer settings are kept at their default values: a 32 bit timer with a period of 1 second and custom preset where all register options are enabled. Interface to the SD-CARD is implemented through four I/O lines: a clock, control and two data lines. All modules memory blocks are internal to the FPGA and are implemented using M4K blocks. Finally, to measure the time spent by software functions, a performance counter is used; however, this timer works in conjunction with the system timer module, and it requires changing its settings such that the timer now has a period of 1μs with a Full Featured register set.

4.2.2. Custom Peripherals

User-built modules integrated into the NIOS II based processor systems are called custom peripherals, they can be used to implement a certain function in hardware as means of hardware acceleration modules. They can take benefit of parallel implementation the hardware can offer for performance-critical scenarios. Moreover, they allow for the added advantage that the NIOS II processor is free to perform other functions in parallel while the custom peripheral operates on data. In our design, the Feature Extraction module and Recognition module are such examples of peripheral modules. Similar to standard peripherals, these modules can have certain control registers to which the processor could read and write to manage their operation.

The OCR peripherals need extensive access to the memory modules, for they need to read the image data, features and weights. One possible implementation is to use the Programmed Input Output (PIO) mode where the NIOS II processor accesses the memory address space on behalf of the modules and performs data transfers to or from the custom peripheral through interface registers, this approach, however, is slow and requires module dependency on the existence of such a processor. Instead, we opted for a design in which the OCR modules could directly access memory on their own; this is accomplished through the Avalon Bus System Interface.

In fact, all standard peripherals are interconnected with each other and the processor through the Avalon Interface, which is a set of defined interfaces for use in both high-speed streaming and memory-mapped applications. There are six different interface types, of which we used the Avalon Memory Mapped Interface (Avalon-MM) which is an address-based read/write interface typical of master–slave connections. For other types of interfaces the reader is referred to (Altera, 2010, Avalon Bus Specifications). Since the Feature Extraction and Recognition Modules need access memory directly, the two have been designed to have an Avalon master interface, and since both need be accessed for debugging purposes as well as internal control and status registers access by the NIOS II processor, both have been designed to have their own Avalon slave interface with their own internal address space. By having both Avalon master and slave interfaces, the feature extraction and recognition modules could communicate directly with each other or through memory. Figure 30 shows the system interconnection fabric and the Avalon interface architecture.

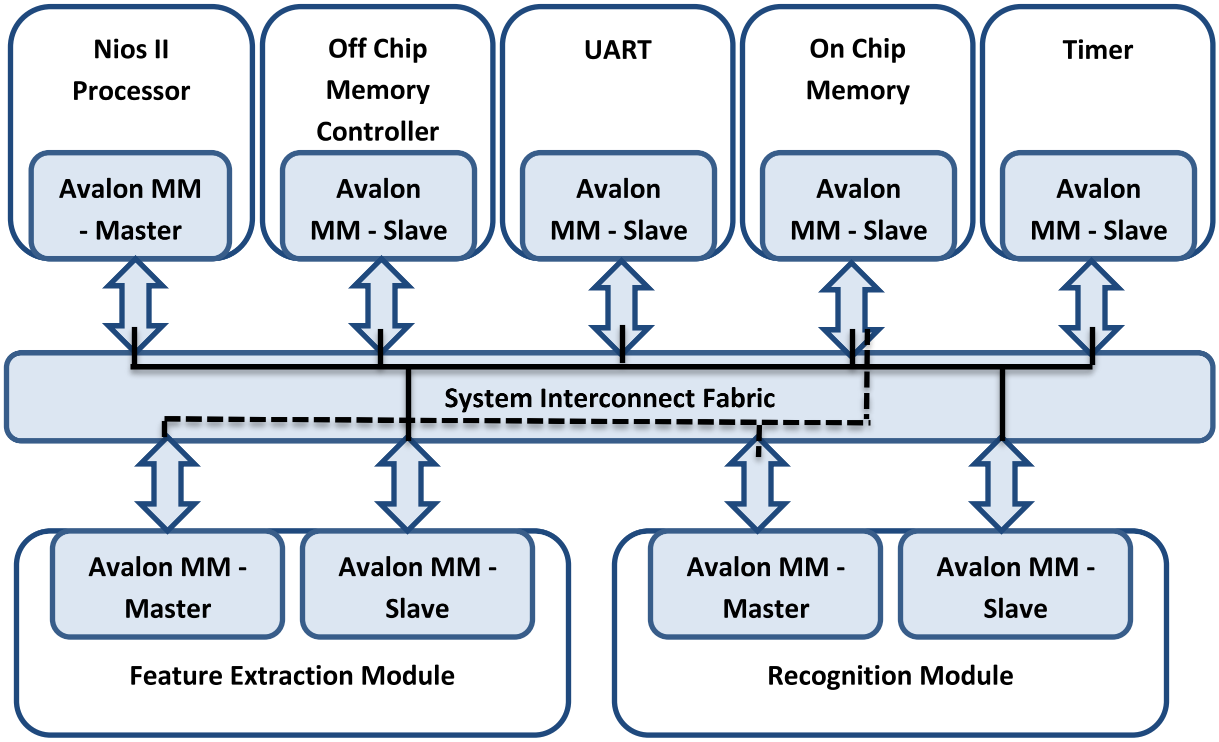


Figure 30 - System intercommunication Graph. Avalon-MM Master interfaces allows for the communication with Avalon-MM slave interfaces. The straight black line represents the modules the NIOS II processor communicates with. The dashed line represents the modules which the OCR modules communicate with.

The Avalon bus system facilitates system design by handling multiple master accesses through arbitration. Even though a DMA controller could provide the similar functionality, its application has been abandoned in favor of providing the custom modules with their own master interfaces, for usually the DMA is controlled by the system processor through the processor's own Avalon master interface, thus is due to our preference to design fully independent OCR modules to serve as IP cores on their own without the need to rely on external modules and/or processors. Also, since the OCR stages need work on different data sets at specific instances, extra communication overhead between the modules and the processor is required to coordinate memory access through DMA to provide the modules with the required data, suggesting more complexity. In addition, the DMA controller can only accommodate quadwords (128 bits) widths transfers (Altera, 2009, Altera Quartus II

9.1 Handbook) while the Avalon interface can accommodate up to 1024 bits, this last feature was used in one of the several recognition module implementations, in which the memory interface of the internal on-chip weights memory was set to this maximum and consequently up to 64 16-bit features were read in one cycle offering the highest speedup. Figure 31 shows the typical signals used in Avalon MM interface.

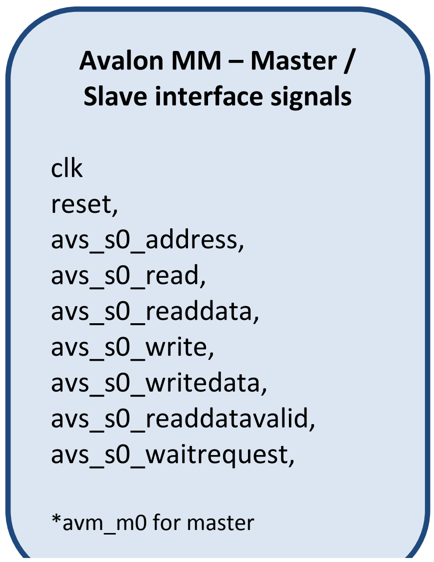


Figure 31 - Typical Avalon MM signals

4.3. Feature Extraction Module Implementation

As concluded in the preceding chapter, the density algorithm was the most suitable for our proposed OCR system implementation, and thereof this subsection will present its hardware implementation as described in section 3.4.2.

Once the image transfer is through, the NIOS II processor writes the internal control register of the feature extraction module, this being checked at every clocked cycle serves as the enable signal for the module. The image is divided into 16 blocks with fixed size of 32x32 bits. The starting addresses of the blocks are internally stored inside the module, the

module starts by acquiring the rows of the first block through its memory interface, the initial block address is set, memory transfer signals are adjusted and the value is transferred and saved, the address is adjusted by increments of 32 bytes. After the end of the block rows transfer, the bits of all rows are summed up in two parallel stages, rows 0 -15, and rows 16-31, the features stored are those of the total sum and the upper block half sum. The procedure is repeated for all the remaining blocks, with change of the block address to account for the feature order. All remaining features from feature sets two to four as shown in Figure 8 are computed in parallel in one cycle from the feature set obtained thus far and of the intermediate sums saved. Finally, all features are orderly and sequentially saved in the feature and weights memory block. The control register is reset to flag operation end. Since the module was coded in Verilog behavioral modeling, it is easier to describe implementation and workflow details in a flowchart format; Figure 32 illustrates the simplified workings of the feature extraction module while Figure 33 through Figure 35 show multiple wave diagrams illustrating the functionality of the module.

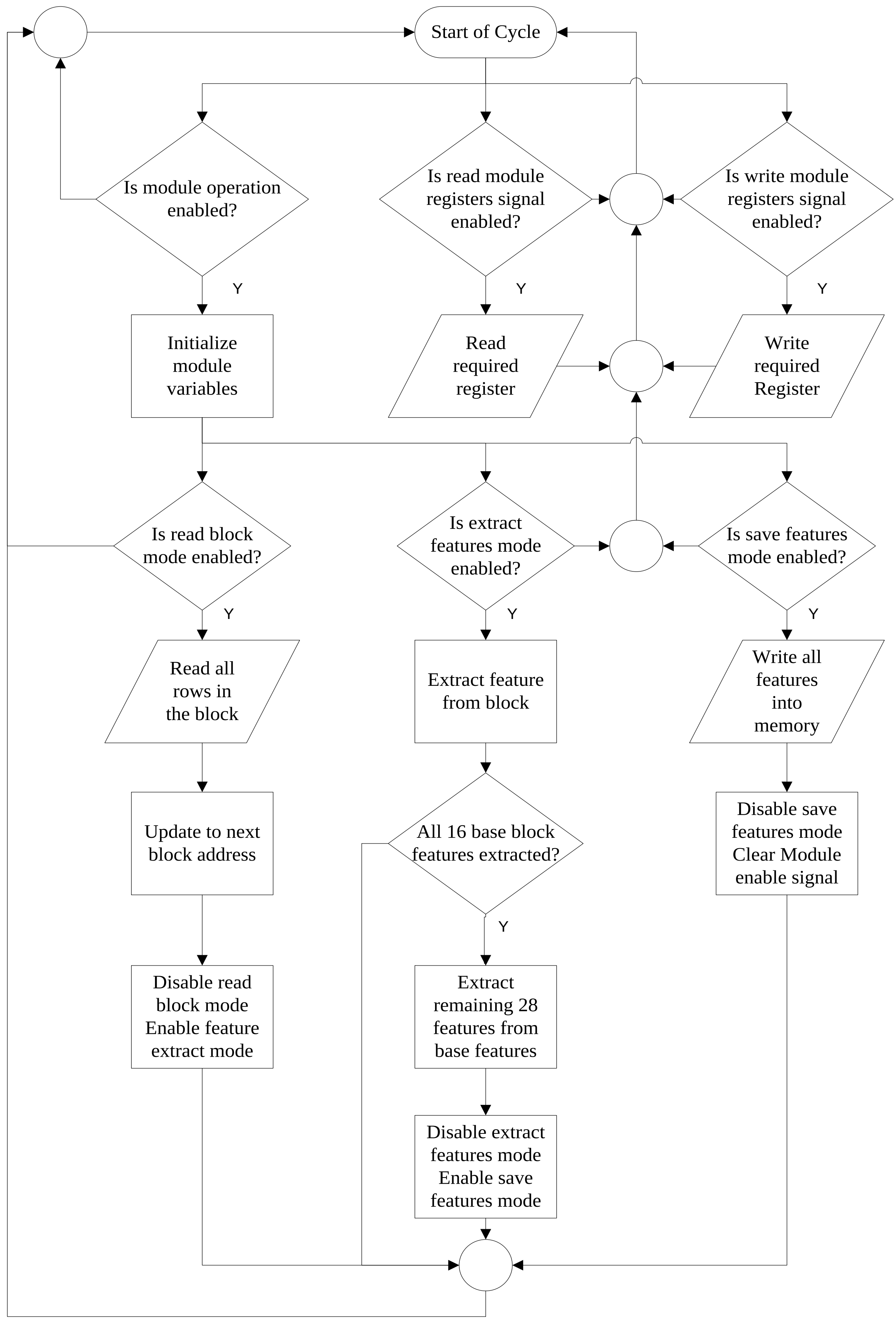


Figure 32 - Feature Extraction module flow diagram

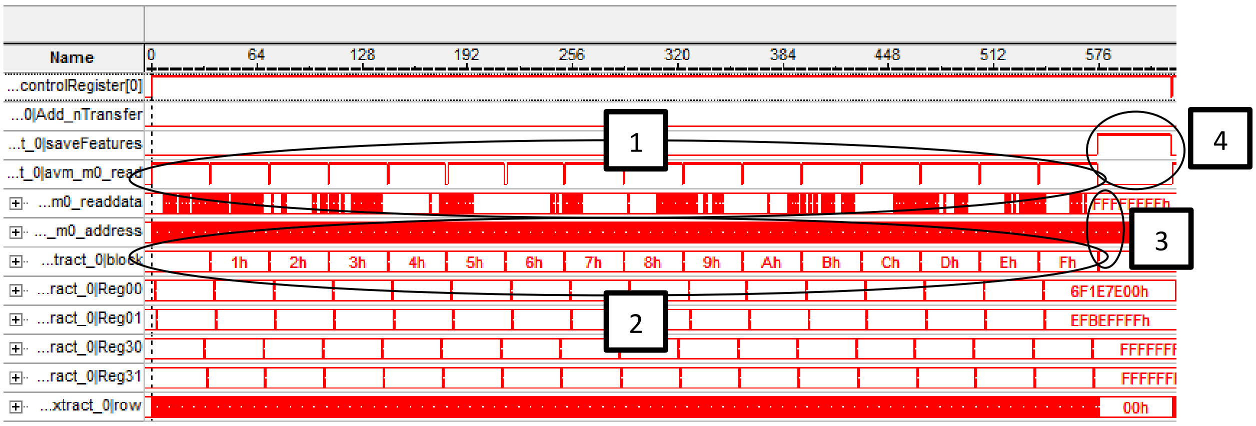


Figure 33 - Feature Extraction Phase Wave Diagram – Reading Image and Extracting Features as seen on the Signal Tap Logic analyzer. (1) As long as the read signal is set, the image is being read row by row in every block, when the signal is reset, a feature is being computed. (2) The module extracts features for the base 16 blocks. (3) All remaining 28 features are extracted in parallel in one cycle time from base features (4) The saveFeatures signal is asserted after all features have been computed, features are stored in on-chip memory.

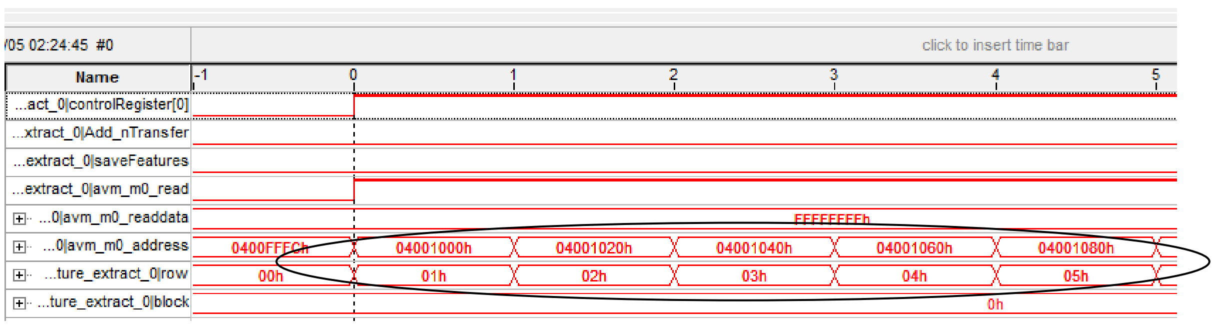


Figure 34 - Feature Extraction Phase Wave Diagram - Reading Image Rows – A Closer Look. For each row the corresponding address is set, notice that for the first block, addresses are 0x20 wide apart, for we are transferring non sequential rows (all rows are on top of each other), and since the image is 256 bits wide, block rows are 32 bytes apart, thus addresses are adjusted by increments of 0x20.

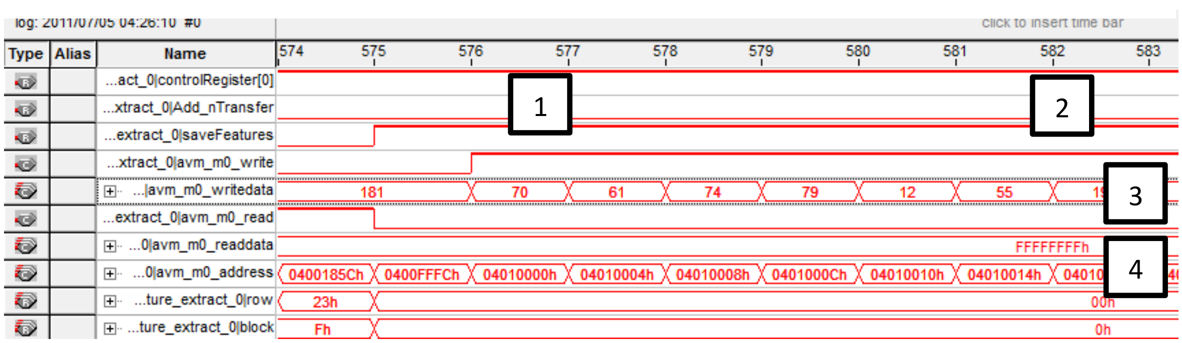


Figure 35 – Feature Extraction Phase Wave Diagram - Saving Features (1) The save feature signal is asserted by the controller. (2) in the next clock cycle, the system asserts the memory write signal . (3) the features are written to the Avalon master datawrite lines. (4) The corresponding write address, the features are sequentially saved.

4.4. Neural Network Recognition Module Implementation

Due to the multiplication-intensive nature of the neural network and the limited logic resources, we adopt an implementation where all the functionality of the network is based on one neuron. Based on our analysis in Chapter 3, we are to use architecture with 44 input neurons, 80 hidden layer neurons and 50 output neurons corresponding to the word set in hand. The total number of multiplications required is thus $44 \times 80 + 80 \times 50 = 7520$, since all our numbers are represented in 16 bit fixed point format, and the cost of 16x16 signed integer multiplier is 624 LEs, clearly a fully parallel implementation for all neural network nodes is infeasible. Moreover, even if one neuron is considered, parallelizing the multiplication process is quite logic consuming with an estimated 49920 LEs for an implementation of one output neuron occupying around ~73% of chip logic resources.

Consequently, we have considered using the embedded multipliers (a specialized fast 9x9 or 18x18 multipliers built in many Altera FPGAs with their own area and resources excluded from those available to the designer in the logic fabric), the Cyclone II EP2C70 FPGA provides 300 embedded 9×9 multipliers, or the same multipliers can be reused as 18x18 blocks offering 150 multipliers. Such multipliers are quite fast for they can provide results in one cycle time. For our 16×16 operations, a total of 44 + 80 multipliers are needed. Therefore, the hardware architecture we adopt herein is to design a serial neural network yet with all arithmetic operations within the neuron done in parallel.

Initially, the first stage of the neural network is to acquire the 44 features stored in the features and weights on chip memory, all inputs of the neural network are to be normalized to the range of -1 and 1 as required in order for the network to work correctly. This will be

done in the same cycle as the features are being acquired saving time. The equation for normalization which is the used in MATLAB in the neural network is the one used in the "mapminmax" function:

$$X_{NRM} = (Y_{MAX} - Y_{MIN}) \times \frac{X_{IN} - X_{MIN}}{X_{MAX}} - 1 \tag{16}$$

Where Y_{MAX} = 1, Y_{MIN} = -1

The normalization parameters X_{max} and X_{min} vary for each feature. Yet it has been found that for the image set used in this research that X_{min} = 0 for all features and X_{max} remains as the only unknown. This reduces the formula to $\frac{2X}{X_{MAX}}$ -1 which can be rewritten as $X \times P$ *-1* where P = $\frac{2}{X_{MAX}}$. However, this formula needs readjusting to account for the fixed point format used; a similar formula which directly results in 16-bit signed fixed point value is:

$$X_{NRM} = X_{IN} \times Round\left(\frac{2^{N+1} - 1}{X_{MAX}}\right) - 2^{N} \tag{17}$$

where N is the number of the fixed point fractions, and rounding is to the nearest integer, since our implementation uses 9 bit fractions, the above equation changes to:

$$X_{NRM} = X_{IN} \times Round\left(\frac{1023}{X_{MAX}}\right) - 512 \tag{18}$$

However, to reduce the error of rounding, the equation is adjusted to

$$X_{NRM} = \left(X_{IN} \times Round\left(\frac{1023}{X_{MAX}} \times 2^{M}\right)\right) \gg M - 512 \tag{19}$$

The maximum value of M was found to be 14 given that the maximum size of multiplication inputs is 18 constrained by the width of the embedded multiplier, and by the minimum value between all values of X_{max} of all features, so the final normalization equation used is:

$$X_{NRM} = \left(X_{IN} \times \text{Round}\left(\frac{1023}{X_{max}} \times 16384\right)\right) >> 14 - 512 \quad (20)$$

All the values of $Round\left(\frac{1023}{X_{max}} \times 16384\right)$ are pre-computed in advance and stored internally inside the module. Normalization is applied in the same cycle the value is retrieved from memory requiring no extra overhead.

So for every neuron in the hidden layer, all the associated 44 weights are retrieved from memory, normalized and multiplied by the input features using the embedded multipliers in parallel, the results are adjusted to account for the fact that we are using 16 bit fixed point multiplication by discarding the least significant 9 bits and only retaining bits 9 through 24, the results of all multiplications are summed.

The hidden layer uses the tansig transfer function, many implementations have been considered to approximate its nonlinearity as described in section 2.3.2, mainly piecewise linear approximation (PWL) and look up tables (LUT). We have chosen to approximate the tansig function using LUT to save on logic resources for the internal memory resources are plentiful in comparison. Moreover, one could make use of the odd symmetry of the function to cut the table size to half. Our investigation has shown that the values from 0 to 3.875 captures all the nonlinearity of the function while larger values than 3.875 map to 1. The negative values have the exact dynamic range of the tansig function but in negative.

Due to function saturation, we were only concerned to use a table which captures the input range of -3.875 to + 3.875, with a 16 bit signed fixed point representation with 9 bits fraction, and thus number increments by 1/512, a table size of 3970 elements (7940 bytes) is implemented. Moreover, when the input value falls beyond the specified range, that is larger than 3.875 or less than -3.875, the appropriate tansig result is returned.

All this can be easily achieved based on the sign bit value; the sign of the resultant summations for all hidden layer neurons is saved, and the sums which are negative are 2's complemented. The now positive results are added to the LUT base address to retrieve the approximated tansig value.

Now, all retrieved values are either retained as is or 2's complemented to account for the original sign of the summed result of the neuron. These new values are the inputs to the output layer neurons, which in the same manner retrieves each neuron associated weights, now numbered 80, multiplied with the layer inputs, adjusted as before and summed. Since the output layer transfer function is linear with $y = x$, the resultant sums for every neuron is the final outputs of the recognition phase. Figure 34 illustrates the flow of operation in the neural network based recognition module.

Six different implementations for this module have been analyzed, where the Avalon MM – Master interface width was varied from16, 32 (Default) to 64, 128, 256, 512 and 1024 bits, and adjusted the feature and weights memory bus width to match accordingly. This will allow us to retrieve 4, 8, 16, 32 and 64 feature / weights in one cycle respectively. However, the number of embedded multipliers for normalization purposes will increase accordingly. Only implementations with 64, 128, 256 and 512 bus width aside from the

default one were implemented for the 1024 requires additional 64 embedded multiplier exceeding device capacity, yet its performance was projected mathematically extending the results from the other implementations. Figure 36 illustrates the simplified workings of the recognition module while Figure 37 and Figure 38 show multiple wave diagrams illustrating the functionality of the module.

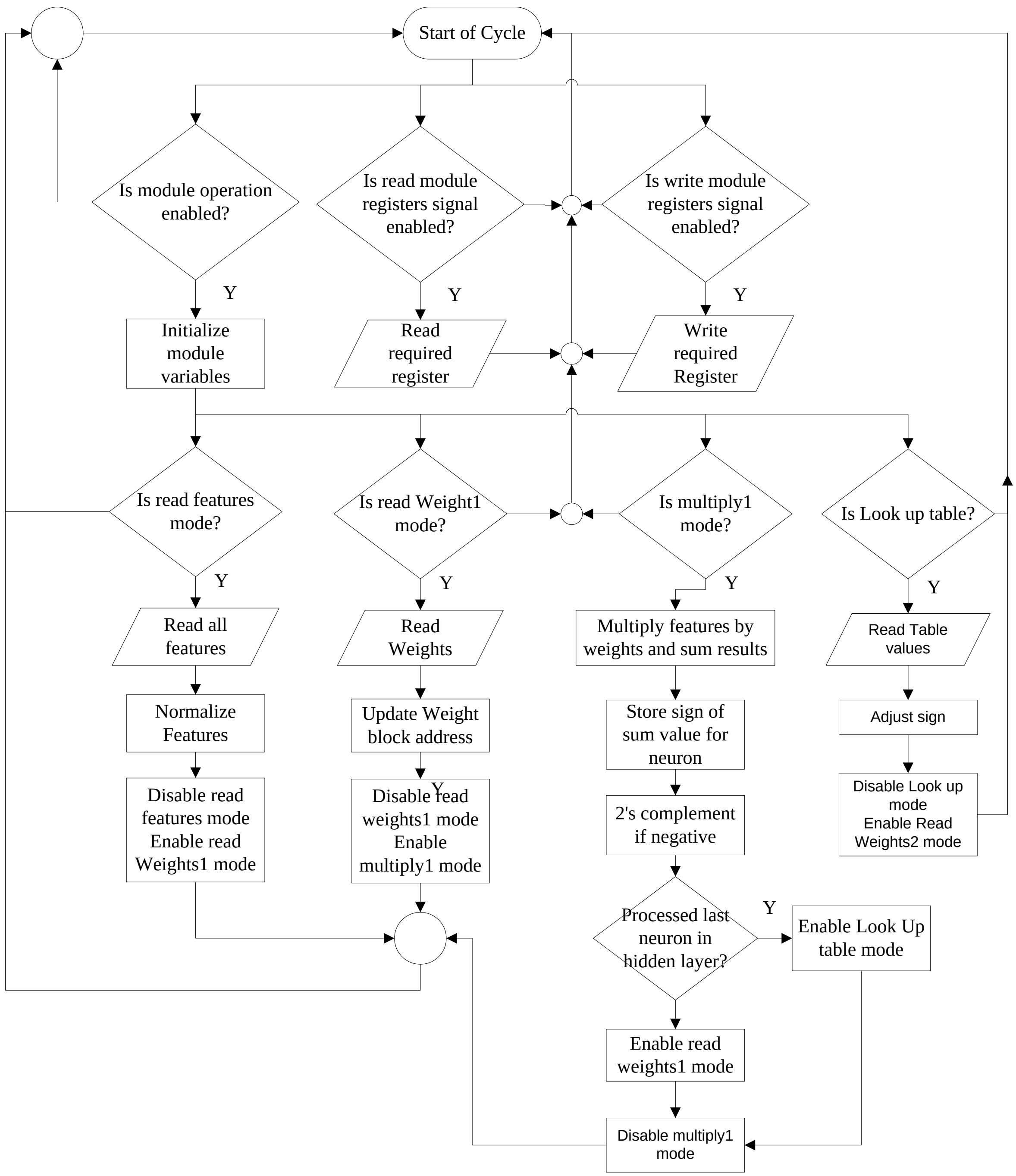


Figure 36 - Neural Network based recognition module flow diagram – Read Weight2 and multiply2 modes are exactly the same as read weights1 and multiply1 albeit the multiply2 step terminates by disabling the module

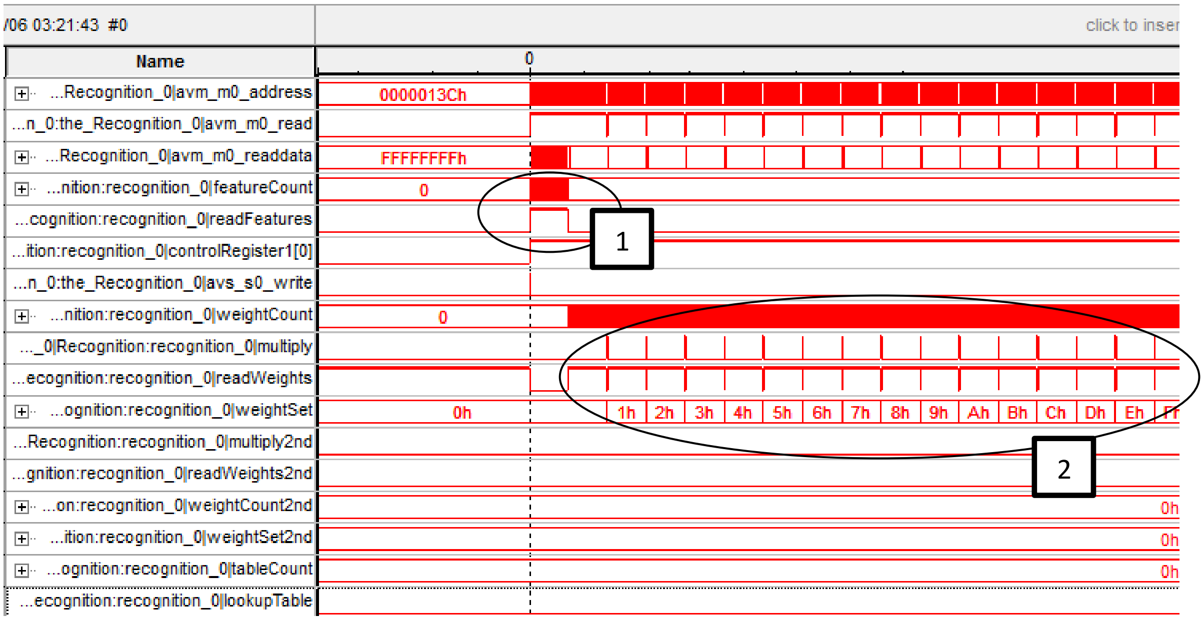


Figure 37 - Recognition Phase Wave Diagram - Features and Hidden Layer (1) Once the control signal for the recognition module is set, it starts reading the features from the features memory, every features is normalized in the same cycle as it is fetched (2) The features are the inputs to every hidden layer neuron where they are now multiplied with their associated weights and summed. The sign of each sum is retained and negative sums and two's complemented before all sums are used to access the "Tansig" look up table.

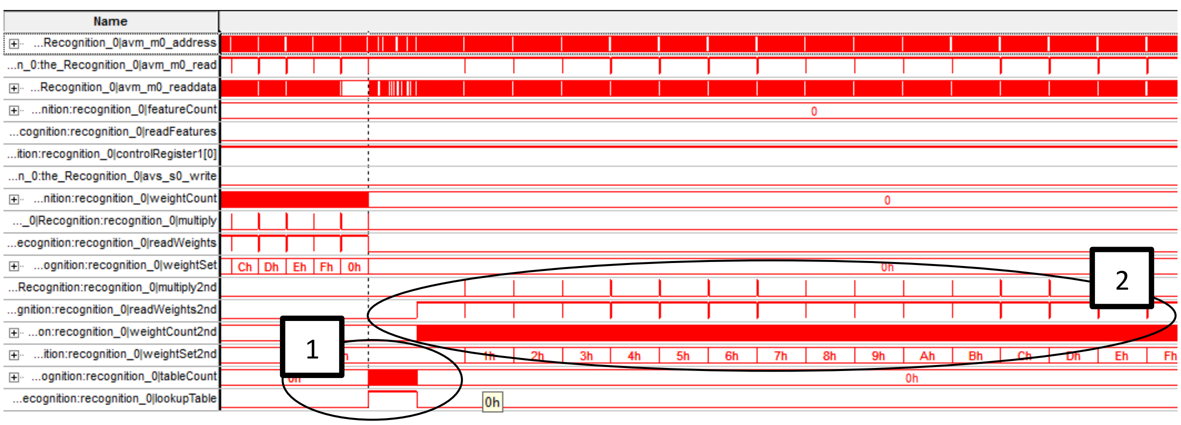


Figure 38 - Recognition Phase Wave Diagram - LUT and Output Layer (1) Once all the hidden layer weights have been multiplied by the features and summed, the "tansig" transfer function is applied through the use of a look up table (2) The values of the LUT, the output of the hidden layer nodes are now inputs to the output layer, they are multiplied by their associated weights in each neuron and summed.

Chapter 5 – Results and Discussion

In the previous chapter, the hardware implementation of the FPGA based OCR system was introduced. This chapter discusses the results of this implementation and its variants where we show the speed up gains, total FPGA resources cost as well accuracy results due to hardware design choices and approximations. MATLAB execution speed is reported based on a system equipped with an Intel core T6600 processor at 2.2GHz, 2MB L2 cache and a 800MHz bus, system memory is 4GB DDR2 with 800MHz bus, on a fresh installation of 64-bit Windows 7 Ultimate version. Simulation was conducted over 100 times for a set of 8699 image for the speed test; averaged results are shown for MATLAB while exact figures are reported for the hardware implementation. Hardware system timing is measured using the SignalTap II logic analyzer IP core, the total number of cycles from the setting of the module's start bit until it is reset is fixed for every image; then the total number of cycles is multiplied by the reciprocal of F_{MAX} to get the total execution time, the time is rounded to the nearest integer and displayed in µs. Accuracy results are based on a test set of 1304 word images (15% of the total image set), this set was not used in the training step.

5.1. Speed up Gains

Figure 39 illustrates the total execution time for the feature extraction stage; FPGA execution time is calculated when feature extraction module run at F_{max} = 100 MHz. It is clearly obvious that the FPGA is faster than the MATLAB implementation.

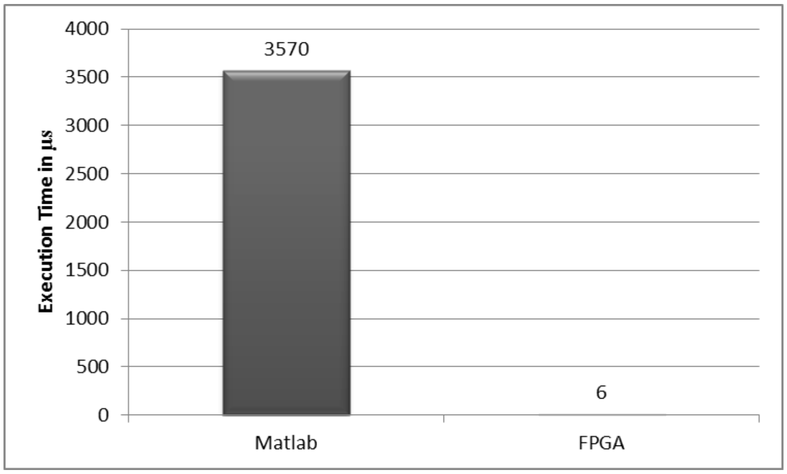


Figure 39 - Execution Time of the feature extraction stage

The recognition module was implemented in three variants each differing in the Avalon MM data bus width and the corresponding features and weights memory width, a system bus width of 16 bits was used, it was expanded to 256 and 512 bits, the maximum choice of 1024 was not tested for the total number of required multipliers for normalization exceeded device capacity. Though 256 and 512 bus widths allow for the transfer of 16 and 32 features/weights in one cycle, we used two implementations of 16 and default 32 bit bus for single data transfer. It was found that maximum system frequency is 45MHz. Moreover, it is found that the recognition system speed can be easily calculated by:

$$\text{Recognition cycles} = ceil\left(16 \times \frac{I_N}{B} + 1\right) + ceil\left(16 \times \frac{I_N}{B} + 1\right) \times H_N + (2 \times H_N + 1) + ceil\left(16 \times \frac{H_N}{B} + 1\right) \times O_N + O_N \quad (21)$$

Where I_N is the number of input features, and equals the number of associated features in hidden layer

H_N is the number of hidden layer nodes

O_N is the number of output layer nodes,

B is the bus width,

And the constant 16 is the size of the fixed point representation of the values.

The first term refers to the total number of cycles needed to read the features, the second term to the number of cycles needed to transfer all hidden layer weights, the third to the number of cycles to access the look up table and perform parallel multiplication, the fourth term to the number of cycles to acquire all the features of the output layer and the final term to the number of cycles need to multiply and sum all output layer neuron inputs and produce a results.

For our architecture where $I_N = 44$, $H_N = 80$ and $O_N = 50$, this equation is expressed as:

$$\textit{Recognition cycles} = \mathrm{ceil}\left(\frac{704}{B} + 1\right) + \mathrm{ceil}\left(\frac{704}{B} + 1\right) \times 80 + 161 + \mathrm{ceil}\left(\frac{1280}{B} + 1\right) \times 50 + 50 \tag{22}$$

Execution time is expressed as:

$$\text{Recognition time} = \text{Recognition cycles} \times \frac{1}{F_{MAX}} \tag{23}$$

Figure 40 shows the execution times of the neural network implementations in MATLAB as well as that of FPGA, the values obtained for the MATLAB implementation are based on the neural network toolbox implementation. Different FPGA implementations with various bus widths are shown. Opposed to the other FPGA implementations, the results for FPGA1024 are numerically computed based on equation 22.

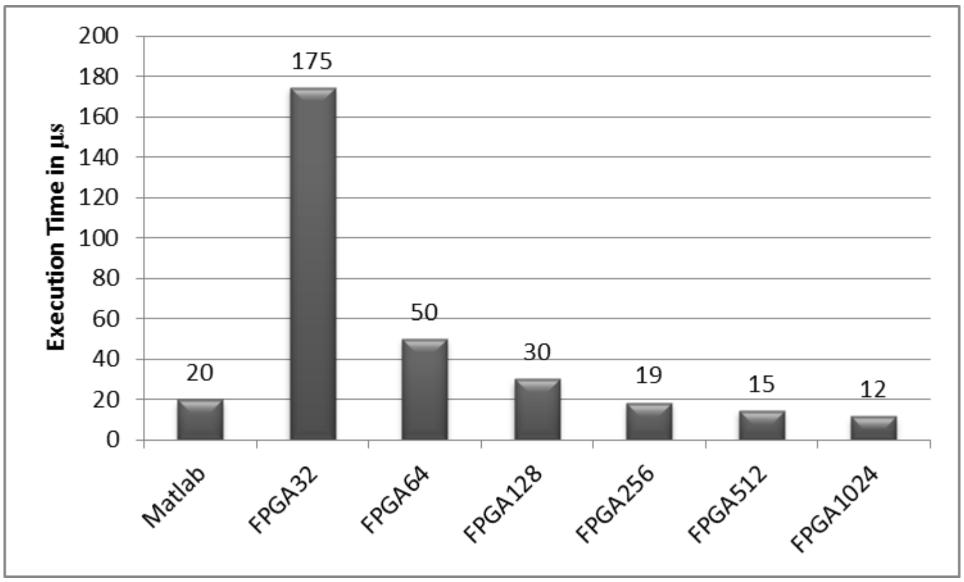


Figure 40 - Execution Time of the recognition stage. FPGAxxxx is an implementation in which xx refers to the bus width in bits.

It is clearly obvious that implementation in which the values are independently acquired offer no speed up advantages over the software approaches, only when the bus size and interface to the features and weights memory is expanded that speed up gains are noticed. Using bus width of size 256 bits offer around 9X increase in performance over one with effective 16 bit data transfers (implementation of default bus of 32, used to carry 16 bit data). Doubling the bus size increases gain by ~30%.

Figure 41 shows the total system execution time. It is obvious that though MATLAB's neural network was quite fast, yet its total execution time is dominated by the totally slow feature extraction stage even though its implementation was based on MATLAB optimized built in functions.

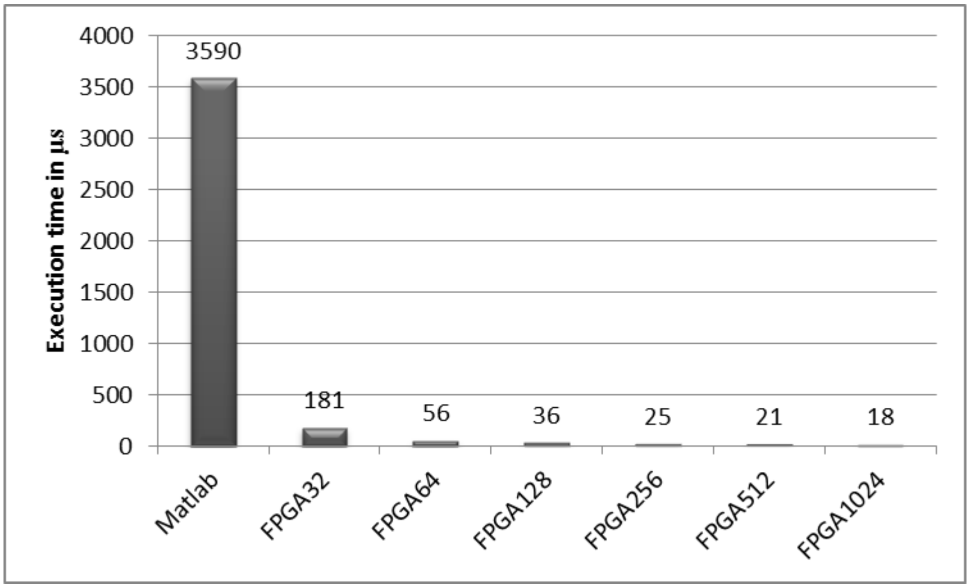


Figure 41 - Total System Execution Time (time (Feature Extraction, Normalization and Recognition)

Final speed tests show that our hardware implementation can be up to ~200 times faster than that of MATLAB. Moreover, when features/weights are loaded into the recognition module in chunks of 64 values, total speed ups of ~9X are recorded over that where one value is acquired in each cycle. In addition, doubling the data bus width in the recognition module from 256 to 512 to 1024, improves total system speed up by a factor of ~20% in each transition.

5.2. Hardware Resources Cost

Figure 42 shows the total system costs. Of the 68416 logic elements, our OCR system occupied 16971 logic elements, around 24.8% of the available logic fabric. The recognition, feature extraction and NIOS II occupy most of the system resources while other supporting modules and interface modules constitute the rest.

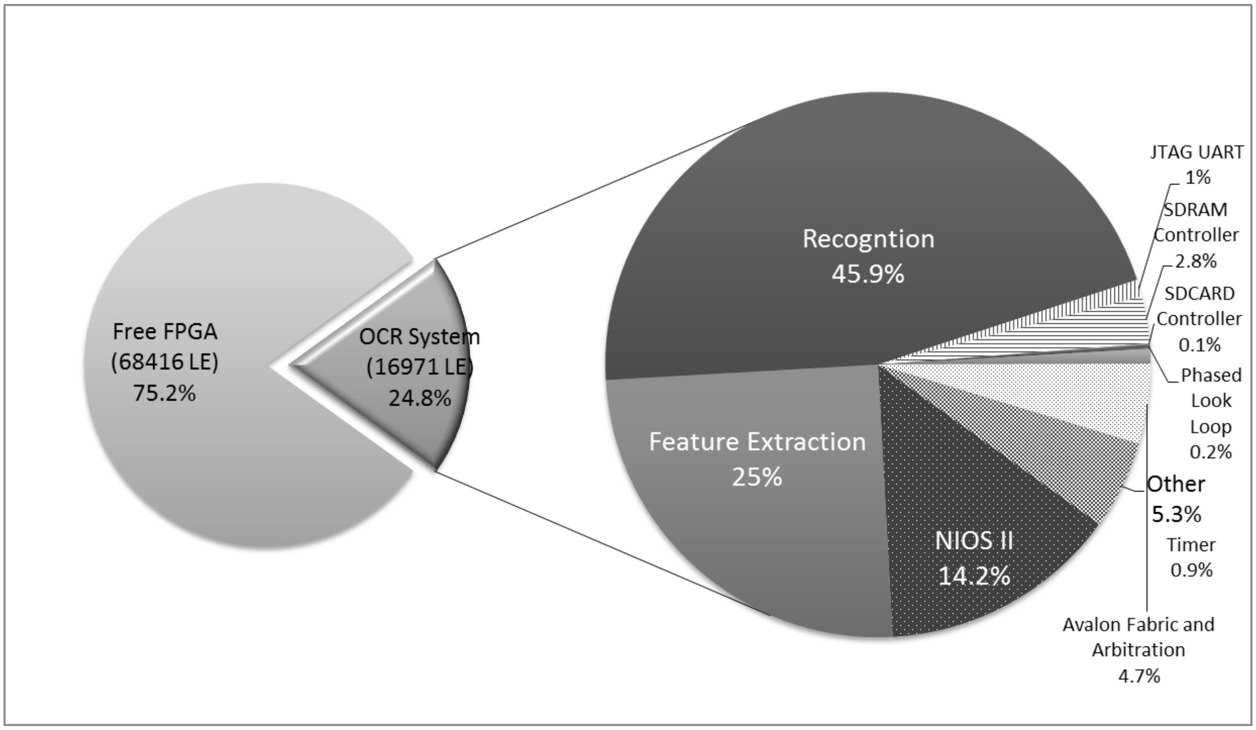


Figure 42 - Total System Cost (Logic Resources) – figures might not sum up to 100% due to rounding

Total system memory usage is nearly quarter that of the total available on chip memory as illustrated in Figure 43, these values are based on using a memory with 16-bit memory width, for when an Avalon MM bus width is increased, and the memory word length is adjusted accordingly, some memory space is wasted due to alignment issues.

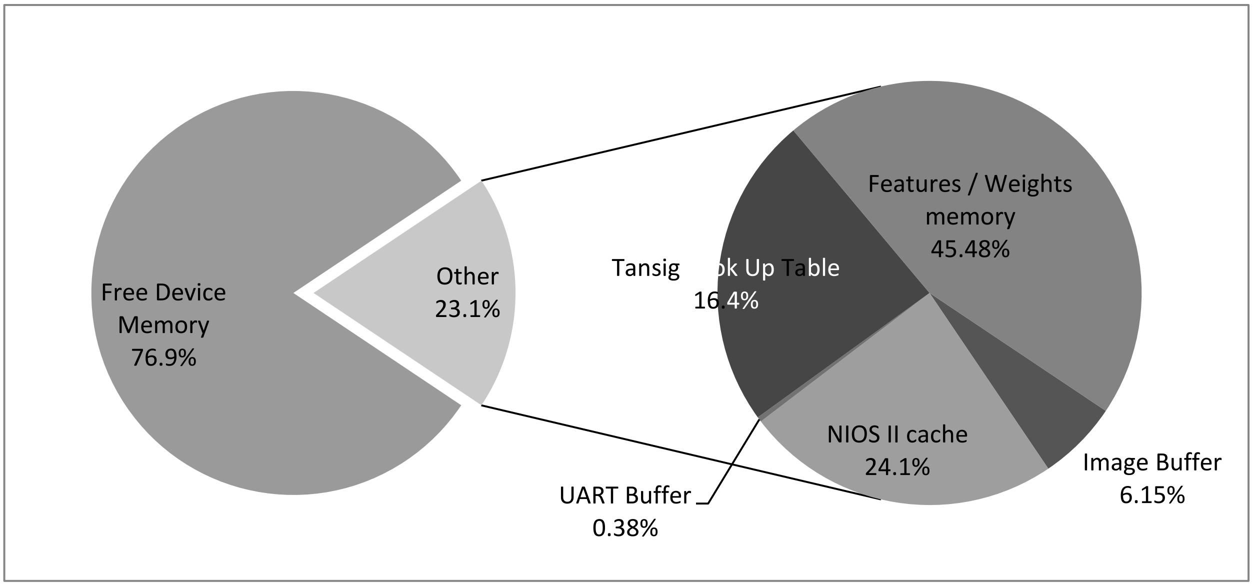


Figure 43 - Total System Cost (On-Chip Memory Resources) – figures might not sum up to 100% due to rounding

To elaborate further, for both the features and the hidden layer weights per neuron, the total number of values to be stored is 44. For a memory width of 256, three memory locations are required to save these features, however, these three locations can withhold 48 values, and the last 4 words are unused due to memory alignment. The same discussion can be extended to the locations needed to save the output layer weights and to account for the other memory location widths. Total wasted memory space due to memory alignment for the different implementations is reported in Table 5.

Table 5 – Wasted memory bytes due to memory alignment – values in bytes

	Total memory	Features space waste	Hidden Layer Weights waste	Output Layer Weights waste	Total wasted memory space	Percentage waste
FPGA256*	15776	4×2	4×80×2	0	**648**	4.11%
FPGA512	19968	20×2	20×80×2	16×50×2	**4840**	24.24%
FPGA1024	23168	20×2	20×80×2	48×50×2	**8040**	34.7%

*FPGA64 and FPGA128 have the same figures as FPGA256

5.3. Recognition Accuracy

In our previous discussion, we have shown that the density algorithm recognition rate was 82.5%; this result was obtained using MATLAB simulations with floating point notation on a test set of 1304 word images (15% of the total image set). On the other hand, our hardware implementation was based on 16-bit signed fixed point format with 9 bit fraction size, and due to this, our representation is less precise with a precision of 1/512. Moreover, we have further had some approximations in the normalization process. Figure 44 shows the recognition results of our system on the same test set above, to analyze the effects of our approximations on accuracy, we had two different implementations for the neural network input normalization stage; the first was implemented in C code and executed by the NIOS II/f processor, the software implementation used equation 16 which is the exact one used in MATLAB, this allowed us to measure the effects of using a fixed point based algorithm over one with a floating point, which resulted in a drop by 1.3% in overall accuracy. In the other implementation, the normalization stage was done in hardware, it

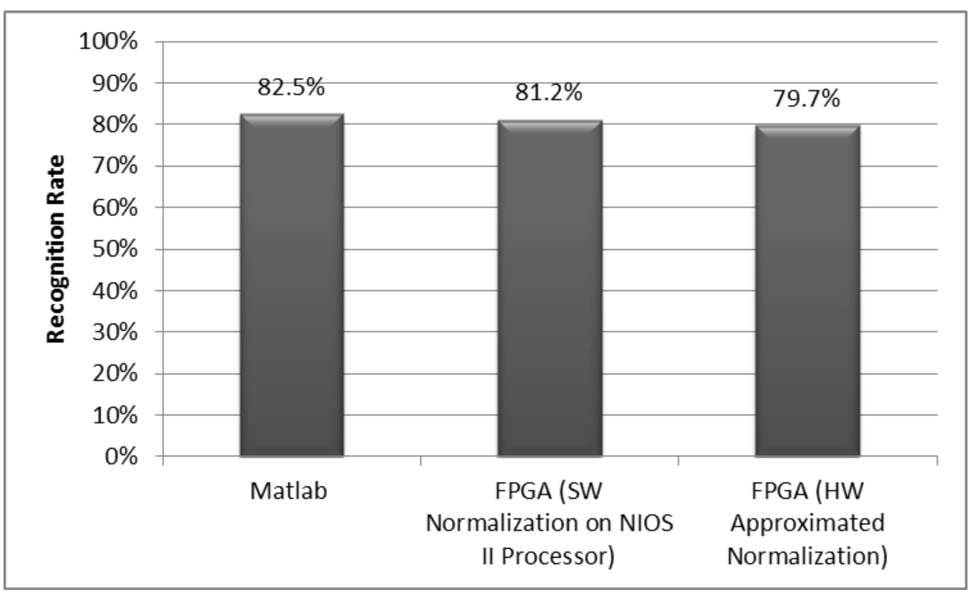


Figure 34 - Recognition rates for software implementations vs hardware based systems

used the approximation in equation 19; the total drop of overall accuracy is 2.8%, which suggests that approximating the normalization process itself has an effect of reducing the accuracy by 1.5%.

Though having the approximation process in software results in slightly higher recognition accuracy, it was not used for it is much slower, on a 100MHz NIOS II/f processor, the total speed of the normalization step alone was about ~1.7ms which would lower the speed of our systems by a 2 orders of magnitude!

Unfortunately, we were unable to compare our results to any of those reported in literature due to the following reasons; initially we have used a subset of the IFN/ENIT database constrained by the topology of the neural network to be implemented and device capacity, most research however reports results over the whole set. Secondly, there exist few modest implementations of OCR on programmable devices for interest has gained momentum only recently and most of these implementations either target the preprocessing steps or are limited to the digit/characters level. Finally, this work is the first of its kind to ever approach the problem of recognizing whole words of the Arabic language using FPGAs based on a holistic paradigm and thus no reference exists for comparison purposes.

Conclusions and Recommendations

The primary goal of this research was to investigate the feasibility of programmable hardware devices, namely FPGAs to serve as a platform for Arabic handwritten OCR system offering the advantage of speeding up some of the OCR algorithms by 20X factor or higher, this goal was sought by selecting a subset of the common feature extraction methods used in handwritten Arabic OCR research, namely DCT, Density, Gradient Masks and Hu moments. Sensitivity analysis was conducted for all algorithms of choice to determine the highest accuracy rates as well as the neural network topology. The analysis was based on recognizing a subset of 50 words of the IFN/ENIT database. Moreover, algorithmic speed analysis based on a MATLAB implementation with results averaged over hundreds of thousands of iterations was conducted. It has been found that DCT had the highest recognition rates at about 88.6% and the fastest speed, closely followed by density (2^{nd} highest speed) and gradient masks while Hu moments fell far behind either in accuracy or speed. The former three algorithms had no further improvement when the number of hidden neuron exceeded 80.

Hardware cost estimates in terms of logic resources and on-chip memory blocks consumed based on parallel implementations of the algorithms, it was found that DCT was the most resource intensive, followed by Hu moments. Density on the other hand was shown to be the most resource friendly. Efficiency of the algorithms was defined in terms of recognition accuracy and hardware costs, the high accuracy of the DCT algorithm was amortized by its high implementation costs leaving the Density algorithm as the best candidate to be implemented on the reconfigurable fabric. A fully parallel DCT implementation for large H×W images could fit or exceed the capacity of modern FPGAs.

The hardware implementation was based on a NIOS II system in which a soft processor assumed the function of the main system controller as well as the responsibility of image transfer into the on-chip memory from the storage source. The feature extraction and the recognition stages were implemented as separate modules with their own interfaces to memory for faster access and module independency. The feature extraction stage was responsible to fully extract the density features from all the 44 defined regions and store them in their respective memory space. The recognition module, a neural network with a 44/80/50 topology was implemented, due to the limited resources of the logic fabric, a sequential network was built instead where a single neuron assumes the role of all the neurons in the network. However, all arithmetic computations within the neuron were performed in parallel; that is all the multiplication operations in between neuron inputs and weights were carried out in parallel.

All neural network inputs are normalized to the range of -1 and 1 as the features were acquired from memory, furthermore, all values were to be in the 16-bit signed fixed point format (S/I/F = 1/6/9) for this will allow us to use the embedded integer multipliers (DSP Blocks) without any further logic costs. Four variations of the recognition module were implemented were each had the ability to transfer features / weights in different chunk sizes. It has been found that the FPGA’s feature extraction module had over 600X speed up factor over the MATLAB algorithm; whereas a modest speed up of ~1.67X over MATLAB was obtained in the recognition module and only when the data was collectively transferred. However, total system speed up was ~200X. Though recognition rate accuracy was 82.5% for MATLAB’s implementation, the effect of all hardware approximations was not that significant, and the sacrifice of some accuracy was minimal in favor of the speed

up gains, accuracy slightly dropped to 81.2% due to the fixed point approximation and further to 79.9% when an approximated normalization equation was used. The total system consumed less than quarter of the Cyclone II EP1C70 device. Based on these results the initial objectives of the research are satisfied.

The main contributions are that this research serves as the first of its kind to have implemented an OCR system for the recognition of Arabic handwritten words based on a holistic approach on FPGAs. We showed speed up gains over the same algorithms when executed in software on a Core 2 Duo processor running at 2.2GHz. This is encouraging for higher speed ups are expected should such a system be implemented in assistive technology or handheld devices where low cost processors such as Intel Atom or ARM are used.

Future Work

This work could be expanded further by analyzing different feature extraction techniques as well as use combined features of different algorithms for higher classification accuracy. Different sets of classifiers could also be investigated or even multi-classifier approaches. This research assumed preprocessed images as the input to the system; several preprocessing techniques could be investigated, such as baseline estimation, thinning or scaling. We were only concerned with the simple holistic approach; the more general segmentation-based model would serve as logical upgrade to the system for the use with unconstrained vocabulary. Moreover, one could make use of the free remaining space by adding extra modules; one such addition is a speech synthesis core, this could serve as a prototype for future assistive technology devices for the Arabic language. We can further

utilize this space to implement other feature extraction techniques and use majority voting techniques on their classification results to achieve higher recognition rates.

References

Abandah, A and Anssari, N. (2009), **Journal of Computer Science**, Vol. 5, No. 3, Mar 2009, 226-232

Abdelazim, H., & Hashish, M., (1988). Arabic Reading Machine. **National Computer Conference**, King Abdul Aziz University

Aburas, A. and Rehiel, S. (2008), JPEG for Arabic Handwritten Character Recognition: Add a Dimension of Application. **Advances in Robotics, Automation and Control**, (Ed.), ISBN: 978-953-7619-16-9.

Abdallah S., Investigation of New Techniques for Face Detection, **Thesis**, Virginia Polytechnic Institute and State University

Abdullah, S. and Marsidi, S. (2008), Digitization Of Arabic Materials In IIUM Library: Challenges And Problems. **World Congress of Muslim Librarian & Information Scientists**, Malaysia, 25 – 27 November.

ACDsee website. Last accessed July 14th, 2011. **http://store.acdsee.com/store/acd/en_US/DisplayProductDetailsPage/productID.227092400**

Al-Khateeb, J. Ren, H. Jiang, J. Jianmin, I.Stan S. and El Abed, H. (2008), Word-based handwritten Arabic scripts recognition using DCT features and neural network classifier. **5th International Multi-Conference on Systems, Signals and Devices**, 20-22 July, Amman, 1-5

Al-Hajj, R. Likforman-Sulem L, and Mokbel. C. Combining Slanted-Frame Classifiers for Improved HMM-Based Arabic Handwriting Recognition. T-PAMI, 31.

Ahmadi, A. Abedin, M.D., Kamimura, K. Shirakawa, Y, Mattausch, H.J. and Koide, T. (2004), Associative Memory Based Hardware Design for an OCR System and Prototyoing with FPGA. **Proceedings of Third COE Int'l Workshop on Nanoelectronics for Tera-bit Information Processing**, Hiroshima, Japan, 40-41.

Alma'adeed, S. (2006) Recognition of Off-Line Handwritten Arabic Words Using Neural Network. **Proceedings of the Conference on Geometric Modeling and Imaging: New Trends**, 16-18 August, London, England, 141 – 144

Altera Website. Last accessed on July 14th, 2011. **http://www.altera.com/products/devices/cyclone2/overview/cy2-overview.html** and **http://www.altera.com/products/devices/cyclone2/literature/cy2-literature.jsp**

Agrawal, R. (2010), *FPGA Implementation of DHT Algorithms for Image Compression.* **Thesis**, National Institute of Technology Rourkela

Amin, A. and Mansoor, W. (1997), Recognition of Printed Arabic Text using Neural Networks. **Proceedings of the 4th International Conference on Document Analysis and Recognition**, 18- 20 August, Ulm, GERMANY

Beg, A. (2008), An Efficient Realization of an OCR System Using HDL, **The 2008 World Congress in Computer Science, Computer Engineering, and Applied Computing**, 14-17 July, Las Vegas, Nevada, USA

Beg, A. Faheem, A. and Campbell, P. (2010), Hybrid OCR Techniques for Cursive Script Languages - A Review and Applications. **2nd International Conference on Computational Intelligence, Communication Systems and Networks**, Liverpool, United Kingdom, 28-30 July, 101-105

Bhandari, S.U. Subbaraman, S. Pujari, S.S. and Mahajan, R. (2009), Real Time Video Processing on FPGA Using on the Fly Partial Reconfiguration, **International Conference on Signal Processing Systems**, Singapore, 15-17 May, 244-247

Bhowmik, D. Amavasai, B.P. and Mulroy, T. (2006), Real-time object classification on FPGA using moment invariants and Kohonen neural networks. **5th Chapter Conference on Advances in Cybernetic Systems**, Sheffield, United Kingdom, 7-8 September

Biadsy, F. El-Sana, J. Habash, J. (2006), Online Arabic Handwriting Recognition Using Hidden Markov Models. **The 10th International Workshop on Frontiers of Handwriting Recognition**

Burrow, P. (2004), Arabic Handwriting Recognition, **Thesis**, University of Edinburgh

CAST 2D DCT IP Core datasheet (2010), last accessed July 14th, 2011. Available on **http://www.cast-inc.com/ip-cores/multimedia/dct/cast_dct-a.pdf**

Cheriet, M. (2008). Visual Recognition of Arabic Handwriting: Challenges and New Directions. **Lecture Notes in Computer Science**, 2008, Volume 4768/2008, 1-21

Cho, J. Mirzaei, S. Oberg, J. and Kastner, R. (2009), FPGA-based face detection system using Haar classifiers. **Proceeding of the ACM/SIGDA international symposium on Field Programmable Gate Arrays,** New York, NY, USA

Draper, B. Najjar, W. Böhm, W. Hammes, J. Rinker, B. Ross, C. Chawathe, M. and Bins, J. (2000) Compiling and Optimizing Image Processing Algorithms for FPGA's. **International Workshop on Computer Architectures for Machine Perception (CAMP)**

Draper, B.A., Beveridge, J.R., Böhm, A.P.W., Ross, C., Chawathe, M. (2003), Accelerated image processing on FPGAs**. IEEE Transactions on Image Processing**,1543-1551

Ebrahimpour, R., Davoudi Vahid, R. D. and Nezhad, B.M. (2011), Decision Templates with Gradient based Features for Farsi Handwritten Word Recognition. **International Journal of Hybrid Information Technology**, Vol. 4, No. 1, January, 2011

El-Abed, H. and Margner. V. (2007), Comparison of Different Preprocessing and Feature Extraction Methods for Offline Recognition of Handwritten Arabic Words.. **Proceedings of ICDAR'07**, Vol.2, 974-978

El-Fegh, I. Suliman, Z. Zubi, Elrowayati, and A. El-Mouadib, F. (2009), Handwritten Arabic Words Recognition using Multi Layer Perceptron and Zernik Moments. **Proceedings of the 10th WSEAS International Conference on Evolutionary Computing**, Wisconsin, USA

El-Hajj, R. Likforman-Sulem, L. and Mokbel, C. (2005), Arabic Handwriting Recognition Using Baseline Dependant Features and Hidden Markov Modeling. **Proceedings of ICDAR'05** , 893-897

Elbeheri, G. Mahfoudhi, A, and Everat, J. (2009), Perspectives from the Arab World. **The International Dyslexia Association Perspectives on Language and Literacy**, 9 – 12

Farah, N. and Sellami M. (2005) Artificial Neural Network Fusion: Application to Arabic words recognition. **ESANN 2005**: 151-156

Farrugia, N. Mamalet, F. Roux, S. Fan Yang and Paindavoine, M. (2009), Fast and Robust Face Detection on a Parallel Optimized Architecture Implemented on FPGA, **IEEE Transactions on Circuits and Systems for Video Technology**, Volume: 19 Issue:4, 597 - 602

Fujinaga, T. Miura, K., Noguchi, H. Kawaguchi,H. and Yoshimoto, M. (2009), Parallelized Viterbi Processor for 5,000-Word Large-Vocabulary Real-Time Continuous Speech Recognition FPGA System. **10th Annual Conference of the International Speech Communication Association,** Brighton, United Kingdom, 6-10, September

Graves, A. and Schmidhuber, J. (2009) Offline handwriting recognition with multidimensional recurrent neural networks. **Advances in Neural Information Processing Systems** 21.

Guo Z, Hall R. (1992) Fast fully parallel thinning algorithms. **Computer Vision, Graphics and Image Processing**; 55(3): 317–328

Haraty, R. and Ghaddar, C. (2004) Arabic Text Recognition, **The International Arab Journal of Information Technology**, Vol. 1, No. 2, 156-163

Holley, R. (2009), Analysing and Improving OCR Accuracy in Large Scale Historic Newspaper Digitization Programs. **D-Lib Magazine**, March/April 2009, Volume 15 Number ¾

IFN/ENIT database Readme file 2003 [Internet]. last accessed July 14th, 2011. Available from **http://www.ifnenit.com/update/README.pdf**

Intel Reader Press Kit webpage, last accessed July 14th, 2011, **http://www.intel.com/pressroom/kits/healthcare/reader/**

Irispen product page. last accessed July 14th, 2011, **http://www.irislink.com/c2-1692-189/IRISPen-6--Overview.aspx**

Jang B. and Chin R. (1992) One-Pass Parallel Thinning Analysis, Properties, and Quantitative Evaluation. **IEEE Transactions on Pattern Analysis and Machine Intelligence**; 14(11): 1129–1140

Jones, C. Berry, L. and Stevens, C. (2007). Synthesized speech intelligibility and persuasion: Speech rate and non-native listeners. **Journal of Computer Speech** 21, 641–65

Jyotheswar, J. and Mahapatra, S. (2007), Efficient FPGA implementation of DWT and modified SPIHT for lossless image compression. **Journal of Systems Architecture Volume 53, Issue 7**, July 2007, 369-378

Kanungo, T. Marton, G.A. and Bulbul, O. (1999) OmniPage vs. Sakhr: Paired Model Evaluation of Two Arabic OCR Products. **Proceedings of SPIE Conference on Document Recognition and Retrieval**, San Jose, CA , 27-28 January, 109-120

Kim, G. and Govindaraj, V. (1996),Efficient chain-code-based image manipulation for handwritten word recognition. **Storage and Retrieval for Image and Video Databases**

Khorsheed, M.S. (2002) Off-Line Arabic Character Recognition – A Review, **Pattern Analysis & Applications**, vol.5, 31-4.

Larmagnac J. (1998) Thinning and Line Segmentation by Line Following Technique. **Proceedings of the International Society for Optical Engineers, SPIE**; 3305: 210–219

Lavrenko, V. Rath, T.N. and Manmatha, R. (2004), Holistic Word Recognition for Handwritten Historical Documents. **Proceedings of First International Workshop on Document Image Analysis for Libraries**, 278 – 287

Lee, Y. (2007), A neural network face detector design using bit-width reduced FPU in FPGA. **Thesis**, University of Saskatchewan

Lorigo L. M. and Govindaraju, V. (2006), Offline Arabic handwriting recognition: a survey", **IEEE Trans. On Pattern Analysis and Machine Intelligence**, vol. 28, 712-724.

Margner, V. and El Abed, H. (2009), ICDAR 2009 Arabic Handwriting Recognition Competition, **10th International Conference on Document Analysis and Recognition**, Barcelona, 26-29 July, 1383-1387

Märgner, V. and El Abed, H. [Internet] Arabic Word and Text Recognition -- Current Developments. Available from **http://www.elda.org/medar-conference/pdf/46.pdf**

Mathworks Website, last accessed July 14th, 2011. **http://www.mathworks.com/products/matlab/**

McCready, R. (2000), Real-Time Face Detection on a Configurable Hardware Platform, **Thesis**, University of Toronto, Ontario

Ministry of Education (2010) – Statistics on Students with Reading-related Learning Disability in Jordanian Schools (Unpublished)

Mirzaei, S. (2010), Design Methodologies and Architectures for Digital Signal Processing on FPGAs, **Dissertation**, University of California Santa Barbara

Moradi, M. Pourmina, M. Razzazi, F. (2010), A New Method of FPGA Implementation of Farsi Handwritten Digit Recognition. **European Journal of Scientific Research**, Vol.39, No.3, 309-315.

Noaparast, K. and Broumandnia, A. (2009), Persian Handwritten Word Recognition Using Zernike and Fourier–Mellin Moments. **5th International Conference: Sciences of Electronic Technologies of Information and Telecommunications**, 22-26 March, Tunisia

Omnipage Professional 18 product page, last accessed July 14th, 2011, **http://www.nuance.com/for-business/by-product/omnipage/professional/index.htm**

Paschalakis, S., Lee, P. and Bober, M., (2003), An FPGA System for the High Speed Extraction, Normalization and Classification of Moment Descriptors", ***In Proc. 2003 13th International Conference on Field Programmable Logic and Applications (FPL '03),*** Lisbon, Portugal, 1 – 3 September , **Lecture Notes in Computer Science,** vol. 2778, pp. 543-552

Pechwitz, M. and Maergner, V. (2003), HMM Based Approach for Handwritten Arabic Word Recognition Using the IFN/ENIT – Database. **Proceedings of the Seventh International Conference on Document Analysis and Recognition**, 3-6 August, 890-894

Quartus II IDE webpage, last accessed July 14th, 2011. **http://www.altera.com/products/devices/cyclone2/literature/cy2-literature.jsp**

Razak, Z. Zulkiflee, K. Salleh, R. Yaacob, M. and Tamil. M. (2007), A Real-Time Line Segmentation Algorithm For An Offline Overlapped Handwritten Jawi Character Recognition Chip. **Malaysian Journal of Computer Science**, 20(2): 171-182.

Razak, Z. Zulkiflee, K. Salleh, R. Noor, N.M. and Yaacob, M. (2009). Off-Line Handwritten Jawi Character Segmentation Using Histogram Normalization and Sliding Window Approach For Hardware Implementation. **Malaysian Journal of Computer Science,**Vol. 22, No.1.

ReadIris 12 product page, last accessed July 14th, 2011, **http://www.irislink.com/c2-1584-189/Readiris-12---OCR-Software-------Convert-your-Paper-Documents-into-Editable-Text-.aspx**

Resnikoff, S. Pascolini, D. Etya'ale,D. Kocur, I., Pararajasegaram, R. Pokharel, G.P. and Mariotti, S.P. (2004) Global data on Visual Impairment in the Year 2002. **Bulletin of the World Health Organization**, November 2004, 82 (11), 844 – 851

Rice, S. (1996), Measuring the Accuracy of Page-Reading Systems, **Dissertation**, University of Nevada, Las Vegas

Rosenthal, M. (2006) JPEG Image Compression Using an FPGA, **Thesis**, University of California ,Santa Barbara

Saabni, R., El-Sana, J. (2009), Hierarchical On-line Arabic Handwriting Recognition. **ICDAR'09**, 867-871

Saichand, V. Devi, M, Arumugam, S, and Mohankumar N. (2008), FPGA Realization of Activation Function for Artificial Neural Networks, **Eighth International Conference on Intelligent Systems Design and Applications**, Kaohsiung, 26-28 November, 59 - 164

Saleh, I. Adly, N. and Nagi, M (2005) DAR: A Digital Assets Repository for Library Collections – An Extended Overview**. 9th European Conference on Research and Advanced Technology for Digital Libraries,** Vienna, pp116-127

Sakher OCR software webpage, last accessed July 14th, 2011 [**http://www.sakhr.com/ocr.aspx]**

Srinivasan, S. Zhao, L. Sun, L. Fang, Z. Li, P. Wang T. Iyer, R. Illikkal, R. and Li, D. (2010), Performance Characterization and Acceleration of Optical Character Recognition

on Handheld Platforms. **Proceedings of the IEEE International Symposium on Workload Characterization (IISWC)**, Atlanta, GA, USA, 2-4 December

Su, T.H., Zhang, T.W. Guan, D.J. and Huang, H.J. (2009), Off-line recognition of realistic Chinese hand writing using segmentation-free strategy. **Elsevier Journal of Pattern Recognition**, Volume 42, 167 – 182

Terasic Website. The DE2-70 Board product page. Last accessed July 14th, 2011. Available from **http://www.terasic.com.tw/cgi-bin/page/archive.pl?No=226**

Tommiska M.T. (2003), Efficient digital implementation of the sigmoid function for reprogrammable logic. **IEE Proceedings – Computers and Digital Techniques** 150, number 6, 403-411

Toosizadeh, N. and Eshgi, M. (2005), Design And Implementation of a New Persian Digits OCR algorithm on FPGA Chips. **European Signal Processing (EUSIPCO2005), 13th Conference In European Signal Processing** 4-8 September

Tumeo, A. Regazzoni, F. Palermo, G. Ferrandi, F. and Sciuto, D. (2010), A reconfigurable multiprocessor architecture for a reliable face recognition implementation. **Proceedings of the Conference on Design, Automation and Test in Europe**, Belgium

U.S. Department of Education, (2007), **Twenty-Seventh Annual Report to Congress on the Implementation of the Individuals with Disabilities Education Act**

Yu S, Tsai W. (1990), A new thinning algorithm for grayscale images by the relaxation technique. **Journal of Pattern Recognition; Elsevier**, 23(10): 1067–1076

Zhao, J.; He, J. Wang, P. Li, F. (2009), Research and development of infrared object detection system based on FPGA. **International Symposium on Photoelectronic Detection and Imaging 2009: Advances in Imaging Detectors and Applications**

Zhu, J. and Sutton, P. (2003), FPGA Implementations of Neural Networks – a Survey of a Decade of Progress. **Proceedings of the 13th International Conference on Field Programmable Logic and Applications**, 1062--1066

Altera knowledge base website (2006), "Why did Altera stop providing equivalent gate counts for its APEX II and newer devices?" **http://www.altera.com/support/kdb/solutions/rd02122003_1989.html?GSA_pos=1&WT.oss_r=1&WT.oss=gate%20count**, last accessed on July 2nd, 2011.

Appendix A – Word Listing in the used Image Set

No.	City Name	Zip Code	Mode	No.	City Name	Zip Code	Mode
1	تونس القباضة الأصلية	1000	142	**26**	ربايع سيدي ظاهر	3097	152
2	المنزه ٩	1013	255	**27**	تطاوين ٧ نوفمبر	3263	254
3	حي الإنطلاقة	1064	124	**28**	القلعة الصّغرى	4021	77
4	تونس الشابي	1089	79	**29**	أكّودة	4022	261
5	المرناقيّة ٢٠ مارس	1116	261	**30**	سوسة إبن خلدون	4061	118
6	شوّاط	1134	87	**31**	شمّاخ	4134	263
7	الفحص	1140	124	**32**	الشوامخ	4159	102
8	الشرايع	1251	101	**33**	حاسّي الجربي	4174	85
9	سيدي ابراهيم الزهّار	1273	269	**34**	نقّة	4283	258
10	قرعة النّاظور	1293	105	**35**	أولاد الشّامخ	5120	140
11	الفكّة	1294	128	**36**	المحارزة ١٨	5154	217
12	جبل الرّصاص	2064	108	**37**	نحّال	6051	245
13	المنزه ٦	2091	242	**38**	الحامّة الجنوبيّة	6060	89
14	سيدي أحمد زرّوق	2112	144	**39**	مارث	6080	257
15	سيدي بو بكّر	2114	122	**40**	الدخّانية	6122	264
16	زنّوش	2116	252	**41**	حي الصلاح	6130	103
17	الرقّوبة	2169	85	**42**	أوتيك	7060	101
18	حامّة الجريد	2214	80	**43**	أوتيك الجديدة	7063	143
19	شتاوة صحراوي	2239	232	**44**	بو زرّاعة	7064	108
20	رأس الذراع	2241	269	**45**	منزل بورقيبة النّجاح	7072	136
21	سبعة آبار	2261	242	**46**	اللواتة	7141	260
22	مركز قصّاص	3013	88	**47**	سيدي الظّاهر	8061	269
23	شعّال	3024	285	**48**	الفايض	9112	261
24	الخليج	3063	262	**49**	الرضّاع	9174	286
25	دوّار اللّواته	3077	81	**50**	أولاد حفّور	9180	83

البحث في بناء محرك تعرف فعال في أجهزة التعرف الضوئي على الكلمات العربية باستخدام الشرائح الإلكترونية القابلة للبرمجة

إعداد

أشرف عماد الصّياغ

المشرف

الدكتور غيث علي عبندة

ملخـــــص

يعتبر التعرف الضوئي على الأحرف العربية المكتوبة مجالاً خصباً للأبحاث العلمية منذ سنوات عديدة و ما زال مع وجود مجال كبير للتطوير خاصةّ للنصوص المكتوبة بخط اليد. على الرغم من ذلك، و على العكس من باقي اللغات، تعتبر تطبيقات أنظمة التعرف على الأحرف العربية محدودة المجالات، مقتصرة على بعض البرامج التجارية، بينما امتد مجال التعرف الضوئي على الأحرف للغات الأخرى ليشمل الأجهزة التعليمية المساندة لذوي الاحتياجات الخاصة، انظمة اتمتة المكاتب، و التطبيقات الخاصة للأجهزة المحمولة. نظراً لخصوصية الكلمات العربية و طبيعة كتابتها بشكل متصل، تم تطوير خوارزميات متطورة معقدة للتعامل معها، إلا أن هذا التعقيد كان على حساب سرعة تنفيذ هذه الخوارزميات، الأمر الذي أدى إلى محدودية انتشار انظمة التعرف الضوئي على الأحرف العربية إلى مجالات تطبيقية عديدة. تم استخدام المعدات القابلة للبرمجة مثل الشرائح الإلكترونية القابلة للبرمجة بشكل ناجح في تسريع تطبيقات معالجة الصور و الفيديو، حيث أنها تتيح امكانية المعالجة المتوازية للعديد من خوارزميات معالجة الصور و التعرف الضوئي على الأحرف العربية، موفرة برمجيات أسرع و تطبيقات في أنظمة مضمنة قليلة الاستهلاك للطاقة. في هذه الرسالة، سيتم البحث في بناء مرحلتي استخراج الملامح و التعرف على الكلمات لنظام التعرف الضوئي على الأحرف العربية على مثل الشرائح الإلكترونية القابلة للبرمجة. هذه المراحل مبنية على خوارزميات تستغل مرونة الشرائح الإلكترونية القابلة للبرمجة و قدرتها على توفير المعالجة المتوازية. سيتم تحليل بعض من أكثر خوارزميات استخراج الملامح انتشاراً، و ايراد نسبة دقتها لمجموعة من ٥٠ كلمة من قاعدة بيانات IFN/ENIT، و تقدير كلفة المعدات عند بناء الخوارزميات المقترحة بالاعتماد على طرق المعالجة المتوازية، و في النهاية البناء الفعلي لأفضل خوارزمية بالإضافة الى محرك تعرف مبني على الشبكات العصبونية. في النهاية تم تسريع سرعة النظام أكثر من مائتي مرة بالمقارنة مع سرعته على برمجية الماتلاب من دون التضحية بدقة التعرف و باستخدام أقل من ربع موارد الشرائح الإلكترونية القابلة للبرمجة. في المستقبل، يمكن استغلال الموارد المتبقية لبناء أكثر من طريقة من طرق استخراج

الملامح أو دمجها، و استخدام طريقة التصويت بالأغلبية لمخرجات أنظمة التعرف بغية تحسين دقة النظام.